2018

Using High-Order Prior Belief Predictions in Hierarchical Temporal Memory for Streaming Anomaly Detection

Brody Kutt
bjk4704@rit.edu

MS IN COMPUTING AND INFORMATION SCIENCES
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

MS DEGREE THESIS

The MS degree thesis of Brody Kutt
has been examined and approved by the
thesis committee as satisfactory for the
thesis required for the
MS degree in Computing and Information Sciences

Dr. Dhireesha Kudithipudi, Thesis Advisor

Dr. Zachary Butler, Co-Chair

Dr. Nathan Cahill, Reader

Dr. William Hewlett, External Chair

Date

Using High-Order Prior Belief Predictions in Hierarchical Temporal Memory for Streaming Anomaly Detection

by

Brody Kutt

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences Department
of Computer Science
in partial fulfillment of the requirements for the
Master of Science Degree
at the Rochester Institute of Technology
2018

Abstract

Autonomous streaming anomaly detection can have a significant impact in any domain where continuous, real-time data is common. Often in these domains, datasets are too large or complex to hand label. Algorithms that require expensive global training procedures and large training datasets impose strict demands on data and are accordingly not fit to scale to real-time applications that are noisy and dynamic. Unsupervised algorithms that learn continuously like humans therefore boast increased applicability to these real-world scenarios.

Hierarchical Temporal Memory (HTM) is a biologically constrained theory of machine intelligence inspired by the structure, activity, organization and interaction of pyramidal neurons in the neocortex of the primate brain. At the core of HTM are spatio-temporal learning algorithms that store, learn, recall and predict temporal sequences in an unsupervised and continuous fashion to meet the demands of real-time tasks. Unlike traditional machine learning and deep learning encompassed by the act of complex functional approximation, HTM with the surrounding proposed framework does not require any offline training procedures, any massive stores of training data, any data labels, it does not catastrophically forget previously learned information and it need only make one pass through the temporal data.

Proposed in this thesis is an algorithmic framework built upon HTM for intelligent streaming anomaly detection. Unseen in earlier streaming anomaly detection work, the proposed framework uses high-order prior belief predictions in time in the effort to increase the fault tolerance and complex temporal anomaly detection capabilities of the underlying time-series model. Experimental results suggest that the framework when built upon HTM redefines state-of-the-art performance in a popular streaming anomaly benchmark. Comparative results with and without the framework on several third-party datasets collected from real-world scenarios also show a clear performance benefit. In principle, the proposed framework can be applied to any time-series modeling algorithm capable of producing high-order predictions.

THESIS RELEASE PERMISSION

ROCHESTER INSTITUTE OF TECHNOLOGY

GCCIS Ph.D. PROGRAM IN COMPUTING AND INFORMAITON SCIENCES

Title of Thesis:

Using High-Order Prior Belief Predictions in Hierarchical Temporal Memory for Streaming Anomaly Detection

I, Brody Kutt, hereby grant permission to Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Signature _____

Date

Acknowledgments

I would foremost like to acknowledge Dr. Dhireesha Kudithipudi and the Neuromorphic Artificial Intelligence (Nu.AI) laboratory at the Rochester Institute of Technology for directly making this work possible.

More generally, I am eternally indebted to the people and places that have raised me to this stage of my life in which I have produced this work. They are simply too numerous to list. We humans love to believe in lone heroes but any honest scientist knows that works of thought and intelligence are never truly a one-person job. As the beloved Dr. Albert Einstein once wrote, "A hundred times a day I remind myself that my inner and outer life are based on the labors of other men, living and dead, and that I must exert myself in order to give in the same measure as I have received and am still receiving."

As far as a dedication to any person or entity goes, I'd like to simply leave the reader with this sentiment.

"Science knows no country, because knowledge belongs to humanity, and is the torch which illuminates the world." - Louis Pasteur

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ability to detect both spatial and temporal anomalies is a useful skill in a wide array of domains such as scientific workflow applications [40], economics and mathematical finance [13, 99], many forms of security and surveillance [43, 72, 78], faults and failure detection in industrial systems [102], robotics [44], text mining [75], health and medicine [14, 119], weather forecasting [94], earthquake prediction [86], online gaming [87] and many more. Anomaly detection remains one of the fundamental problems for which AI seeks to provide a solution [20]. Recognizing distinctiveness and novelty can be thought of as a precursor to understanding more complex concepts and abstract collections. Beginning at a very young age [107], detecting anomalous events and objects seems to be one of the most fundamental functions of intelligent behavior and the thinking primate brain. It is no question that anomaly detection is an important problem for advancing artificial intelligence research and real-world applications alike.

1.1 Anomaly Detection by Means of Prediction

As is directly observable in everyday life, a typical pattern of thought for a human to recognize an anomalous event or object requires two stages. Firstly, a definition of normality is learned by direct observation or inferred from the most frequently occurring patterns. Secondly, value judgments are made by the human to determine the subset of events or objects that are most anomalous among the group. These value judgments are based on predictions of what the human expects to observe given the learned definition of normality. If our predictions are broken, feelings of novelty and uniqueness are raised into our conscious awareness. These feelings are what drives a human observer to label a

particular data point or span of time as anomalous. Because of its ubiquity in human life and usefulness for many different tasks, forming accurate predictions in time is the goal of many AI algorithms; especially those which aim to computationally model the primate brain in some sense.

One such theory, discussed further in Section 2.2, is called Hierarchical Temporal Memory (HTM) theory which posits the idea that prediction is the essence of intelligence and it is the activity in which the animal brain, particularly the mammalian neocortex, excels [47]. This contrasts with more traditional philosophies such as the Turing Test which establish behavior as the chief indicator of intelligence [84]. Whichever is closer to the truth, if there is one, is currently more of a philosophical question and is not the subject of this thesis. In any case, we can recognize that the core working principle of prediction lends itself well to time-series anomaly detection tasks. It is a natural application of HTM's sequence learning and prediction algorithms to be able to detect anomalies in time-series data. At a high level, we can compare the predictions of cellular activations arising from learned patterns of normality to the actual cellular activations arising from sensory encoding to derive a sense of predictability of a data point. This can be used to get a glimpse of how much the data point was anticipated from previously learned patterns. This is the central mechanism for anomaly detection with time-series predictive models. Concretely, building a time-series model of a signal to learn its normal sequences and using that model to predict the future is a common theme among algorithms built for temporal anomaly detection [20]. This thesis takes that high-level methodology and re-imagines it in the effort to increase performance. The core idea is to extend independent, instantaneous predictions to overlapping chains of predictions to better illuminate local context and provide for a layer of fault tolerance.

Note that not all temporal anomaly detection algorithms use streaming predictions as the central mechanism for anomaly detection. Other methodologies pose the problem as a static classification problem or a task of determining statistical outliers, for example. More about these other methods is discussed in Section 2.1.3.

1.2 Assumptions

There are a few important assumptions this thesis imposes regarding the general time-series anomaly detection task. Firstly, we assume all incoming data comes in the form of a single time-series scalar signal that is reflecting some measured object or activity. Discussed more in Section 3.1.3; nominal, multi-feature and multi-signal data can easily be accommodated through the intelligent design of data encoders in HTM, if desired. We

say time-series to mean data that is ordered in some way. Except for the first and last entries, each data point appears strictly before and after some other data point in an ordered sequence.

Secondly, to have some expectation to work, the data must follow some form of repeatable patterns and thus be predictable to some degree. The underlying sequence learning and prediction framework used in this thesis is robust to noise and is capable of learning and predicting multiple, simultaneous futures. However, if the data does not follow some semblance of predictability the results will be meaningless.

Thirdly, *any* previously unseen or relatively infrequent behavior in the signal is considered to be rightfully anomalous. Adjustments to the sampling frequency or shifts in the normal level of noise, for instance, would naturally change the signal characteristics and prompt anomalous labels until the continuous learning properties of HTM learn the new signal characteristics.

Lastly, we define spatial anomalies to occur within a very small number of timesteps ($\lesssim 3$ timesteps) and furthermore define temporal anomalies to occur in longer but still relatively short periods of time ($\lesssim 20$ timesteps). Changes in the data that occur spread through larger elapses of time can still be detected with HTM if the deviation from the norm was previously unseen or occurs relatively infrequently. The underlying HTM framework will continuously learn and adjust itself to fit these gradual signal changes as they occur. This behavior is desirable to avoid large false positive rates as data characteristics and seasonality naturally change.

1.3 Research Motivation

This thesis takes the central mechanism of prediction-based anomaly detection a step further and re-imagines it in the effort to increase its detection capabilities and robustness. We use HTM as a case example of the underling time-series modeling algorithm which the proposed framework is built around. The proposed framework can be applied to any time-series modeling algorithm in principle as long as it can produce high-order predictions.

Generally speaking, a lack of local sequence context in prediction error is common to any streaming algorithm that is restricted to only first-order predictions. This is because processing new data after a context shift would naturally alter new predictions to fit the new context. This lack of local contextual awareness reduces overall response of temporal anomaly models. This is shown to occur with the Numenta algorithm without the proposed framework in Section 6.2. In order to capture contextual anomalousness, we need high-order predictions made without any influence of new data. Also characteristic of these

algorithms is a proneness to faults as each first-order prediction is tasked with determining the anomalousness of a timestep all by itself. Shown in Section 6.5.4, using high-order predictions and searching for *accumulations* of prediction error provides a layer of fault tolerance which both avoids false positives and missed real anomalies.

1.4 Research Contributions

In summary, the principle change that the proposed framework introduces is extending independent, instantaneous first-order predictions to overlapping chains of prediction. This is in an attempt to mitigate the limited temporal scope of prediction error calculations and adding a layer of fault tolerance. It is essentially like asking the time-series model for every timestep "starting at this point in time, with no knowledge of the future, what would you expect to happen for the *next handful* of time steps?" *Multiple* predictions for *each* timestep are collected where each prediction was made at a different point in the past which better illuminate local contextual anomalousness and increases fault tolerance. The paradigm of reporting prediction error is also changed from reporting independent first-order error to *accumulations* of error in which only the anomalies with the most evidence to be so survive. This helps to separate normal signal behavior from true anomalies for better detection performance.

We know the columnar organization inside the temporal memory region of HTM models represent knowledge of multiple, simultaneous sequences. The proposed framework attempts to extract and utilize that high-order sequence representation for more sophisticated anomaly detection. This effort is made possible in this thesis through a formulated, implemented and tested mechanism of extracting high-order prior belief (HOPB) predictions out of a single instance of an HTM model. As demonstrated in Section 6.3.2, the HOPB predictions take the previously learned high-order sequences and follow them out in time without the influence of new data. A large collection of experiments in Chapter 6 convey the benefit in performance the full HOPB algorithm provides. Importantly, shown in Section 6.8, we also demonstrate the additional computational overhead of using the HOPB framework to be asymptotically insignificant with respect to the time complexity of the underlying time-series model and manageable in practice. Shown in Section 6.7.2.1, we see HOPB effectively eliminates the need for encoding timestamps in the HTM model. It's worth noting that if the number of HOPB predictions made at each timestep is limited to one, the prediction information is the same as is used in the Numenta algorithm described in [12].

In a general sense, this paradigm of HOPB predictions for anomaly detection can be

applied to any algorithm that is capable of continuous sequence learning and prediction. The underlying time-series algorithm need only be able to produce high-order predictions. Additionally, as HTM itself continues to grow and develop, this framework of HOPB predictions can always be used to increase the scope of context used in calculating prediction error and the confidence of anomaly tags. As the theory continues to improve, the usefulness of HOPB predictions will become greater. HOPB prediction usefulness hinges entirely on the accuracy and reliability of the individual predictions themselves.

1.4.1 Clarifying Note

It is important to note the plurality of "sequences" used above since it is possible that multiple possible futures could follow from a given timestep. Each potential future needs to be accounted for to be able to find the one that matches what took place best and to report the most relevant prediction error accordingly. By design, HTM's internal representation combined with the mechanism of HOPB predictions can represent all high-order possible futures at once in a way that is useful if there is sufficient representational capacity in the network. This is to say the best path is found automatically within the representation itself; no further complex computations need to be done in the presence of arbitrarily many possible paths.

It's also important to understand here that the predictions coming out of the temporal memory algorithm are predictions of internal cellular activations inside the cortical model. It is a prediction of the model's own internal state. Contrary to most usages of the word "prediction" in machine learning, they are not predictions of the next time-series data value. There is a separate methodology for getting scalar or class predictions out of the HTM model which is described in [26].

1.5 Research Goals

There are two principle goals and associated impacts on the AI research community with this thesis enumerated below.

1. First, this research intends to identify and characterize a previously undocumented limitation of the current Numenta anomaly detection algorithm as described in [12] that is characteristic of any algorithm restricted to first-order predictions. This includes hypothesizing the source of the problem and characterizing the effects which it imposes. The details of the latter can be found in Section 6.2.

2. Second, this research intends to introduce, define, build, test and evaluate a novel framework for anomaly detection which is designed to help overcome the limitation exposed in the first goal. This framework is formalized with respect to and built upon HTM as a case example. The details of the framework can be found in Chapter 4 and the results of the experimentation on the framework are in Chapter 6.

3. Third, this research aims to incorporate new datasets to be used for the sake of benchmark comparison and evaluation of streaming anomaly detection algorithms in addition to what is available in the Numenta Anomaly Benchmark (NAB) [64]. These third-party datasets increase the objectivity and credibility of the conclusions drawn in this thesis.

1.6 Thesis Organization

This thesis is organized into seven main chapters. Chapter 2 immediately follows this introduction and is a literature review of the necessary theory and high-level concepts needed to understand this thesis. The two main subjects that are covered are streaming anomaly detection problems and HTM.

Chapter 3 is an exploration of existing algorithms for time-series anomaly detection of all kinds. Special care is taken to explain the current software implementations of HTM theory and the Numenta algorithm which adapts HTM to streaming anomaly detection problems.

Chapter 4 includes a detailed explanation and formulation of the proposed framework for anomaly detection built upon HTM in this thesis. This includes an overview of the motivations and observations that took place that led to this idea as well as concrete algorithms and equations that describe the new framework.

Chapter 5 presents the formal hypothesis which is tested in this thesis. A thorough description of the evaluation methods used to test this hypothesis is provided. This includes custom designed targeted experiments, a description of the Numenta Anomaly Benchmark (NAB) which is used for the sake of breadth of testing as well as detailing additional evaluation datasets from a wide variety of domains used to increase the objectivity and credibility of results.

Chapter 6 is a thorough review of the results obtained by the experiments detailed in Section 5.2. Both the obtained results and a discussion of the results are reported here. Possible explanations for the result of each experiment are given.

Chapter 7 is a summary of the results obtained in Chapter 6 and the conclusions that can be drawn from them. Usage advice and deployment strategies are provided for

implementing the system into real-world applications. Additionally, possible avenues for future work are detailed here for readers interested in research with HTM and anomaly detection.

1.6.1 How to Read This Thesis

This thesis is not necessarily intended to be read linearly. One may benefit to read Chapter 7 first to understand the purpose of the research and the results obtained. If you've never heard of HTM or streaming anomaly detection before, you may benefit from reading Chapter 2 and Chapter 3 next. What is most important is to understand the high level concepts of HTM models instead of the low level algorithmic details. Only once you feel comfortable with the high-level concepts and design of HTM, reading Chapter 4 will then help you understand how we define and use HOPB with HTM as the underlying time-series modeling algorithm. Then reading Chapter 5 and Chapter 6 will help you understand the purpose of each experiment performed and the detailed results of each respectively.

Chapter 2

Background

To understand the contribution of this thesis, it is necessary to understand two things. First is the problem of streaming anomaly detection and the current solutions that exist. Second is to understand is the neurobiological theory known as Hierarchical Temporal Memory (HTM). I will provide the necessary information to get up to speed with these two topics here.

2.1 Anomaly Detection

An anomaly can be defined as any pattern that significantly diverges from an established and recognizable norm to differentiate it from the whole. These non-conforming patterns are also known as outliers, exceptions, discordant observations, aberrations or peculiarities across different domains [20]. Of all the nomenclature and domain specific jargon, anomalies and outliers tend to be the most commonly used and are sometimes used interchangeably. Anomalies are of special interest to humans because they typically contain interesting and actionable information in the signal. Dating back as early as the 19^{th} century, the automatic detection of anomalies is a heavily studied problem [20, 74]. Concretely, with respect to a specific dataset, the problem can be defined as automatically determining which data points correspond to anomalies and which don't. There have been many approaches developed in the past that approach the problem from both an unsupervised and supervised manner. The unsupervised approach, such as HTM's approach, typically relies upon sequence memory and prediction, clustering or lightweight statistical modeling. The supervised approach typically involves training a classifier to learn divisions in the feature space that are meaningful for determining the anomalous status for new data points.

Instead of detailing every theoretical framework and algorithm that has ever been developed for anomaly detection, I will give a high-level overview of the possible classes in which these methods can be categorized. These classes give insight into the kinds of problems they are applicable toward and the way they search for anomalies in general. In addition, I will describe in detail two broad categories of anomalies one can find reflected in data. These different kinds of anomalies are important to understand as they often require different strategies to detect them.

2.1.1 Types of Anomalies

There are two main types of anomalies which I will describe here: spatial and temporal anomalies. The distinction has been made because, in general, different algorithms which employ different kinds of anomaly detection strategies are often suited to detect only one or both kinds of anomalies. It often requires different algorithmic strategies to be able to detect spatial versus temporal anomalies reliably. The reason for this is their differing visibility characteristics which are described below.

2.1.1.1 Spatial Anomalies

Spatial anomalies, also sometimes known as point anomalies, are any anomalies that can be discovered regardless of the time that it occurs and are generally composed of only one or very few data points [20]. These kinds of anomalies do not need to be organized in a time-series to be identifiable although they still can be. These kinds of anomalies are generally considered to be easier to detect because they do not require any sequence memory. Relatively simplistic measures such as standard statistical measures and models on the data is often all that is needed to detect spatial anomalies reliably.

For example, consider this unordered collection of data points {1.4, 3, 2.5, 1.1, 2, 3.2, 25.7, 2.5, 1.7}. Without any knowledge of where this data came from or in what order these data points were captured, we can immediately tell that 25.7 is significantly unlike the rest of the data points and thus anomalous. It wouldn't matter if we shuffled the data points in any order. In time-series data, a spatial anomaly may be defined as such because of the short-term, local deviation from what is normal.

A real world example of a spatial anomaly could be an unanticipated sharp increase in a CPU's temperature to 60°C while a normal operating baseline of 45°C had been well established. This kind of anomaly might signify a broken fan and should be flagged as an anomaly so that remedial action can be taken.

It is important to note that not all deviations from the norm in the spatial dimension

are anomalies. Each individual source of data and data collection method will have a natural distribution of noise associated with it. It is an important job of any anomaly detection system to learn and account for this distribution of noise to not accidentally label benign noise as anomalies. Events of spatial deviation from the norm are only to raise an alarm when the magnitude of deviation is not normal *with respect to* the natural distribution of noise.

2.1.1.2 Temporal Anomalies

Temporal anomalies are any anomaly that is suspect only when considering its relative temporal context and occur during an elapse of time. These kinds of anomalies may span multiple data points and may be or not be in a certain order to qualify as anomalous. Naturally, these kinds of anomalies are exclusively explored in time-series data. These kinds of anomalies are also sometimes known as subsequence-based or behavioral anomalies [20, 22, 97]. Temporal anomalies are sometimes furthermore broken down into subgroups of collective and contextual anomalies. Although they both fall under the same umbrella of temporal anomalies, we can make a distinction between contextual and collective anomalies to better understand the kinds of algorithmic properties needed to detect them.

Contextual Anomalies A contextual anomaly is a sequence of data points which may be expected when present in one context but anomalous for another. For example, a spike in energy usage in a building during the night when it is supposed to be shut down might be considered anomalous but the same spike during peak hours would be considered normal. The difficulty in detecting contextual anomalies lies in its seeming belongingness when considered in isolation. They can easily fool any algorithm that has no concept of high-order sequence context. Implied by name, an algorithm that is fit to detect contextual anomalies must have some sense of sequence context. Changepoints (places where the data deviates from what is expected with respect to temporal patterns in a sustained, non-instantaneous way) would qualify as contextual anomalies with this set of definitions. The data is only detectably anomalous with respect to the temporal context of the current sequential patterns.

Collective Anomalies A collective anomaly is a group of related data points that are not necessarily anomalous when considered in isolation but are anomalous when considered together. For example, a sequence of otherwise normal data points occurring in the wrong order than what is normally seen qualifies as a collectively anomalous event. A col-

lective anomaly may also constitute a group of only mildly spatially deviant data points which by themselves may rightfully be considered benign noise but are of explicit concern when seen directly adjacent of each other. The difficulty in detecting collective anomalies lies in the subtlety of its expression with respect to the level of natural variability of occurring patterns. An algorithm fit to detect collective anomalies must have some sense of accumulated error over time to capture the temporal aspect of the anomaly as well as the spatial aspect.

Additional Considerations Beside the classification of either collective or contextual, a temporal anomaly might also be characterized by additional information. For example, your signal might exhibit seasonality. This is the observation of the set of patterns and distributions present in the signal changing in regular intervals over relatively long periods of elapsed time. An example of a signal with natural seasonality could be daily average temperature readings over the course of several years in a location that has distinct summer and winter weather patterns. Alternatively, your signal might be associated with cyclic patterns that do not have a fixed period length. An example of this could be consumer patterns with the erratic gradual rises and falls of the economy. If they've been seen before, these kinds of signal changes are not typically reflective of an anomaly. Naturally, anomalous behavior that occurs over especially long periods of time requires deeper capacity for sequence memory.

Real-world signals frequently undergo global changes. For example, consider the placement of electrode patches while performing an Electrocardiogram test. While the source pattern of the heartbeat remains the same, different orientations of the electrodes will lead to unique signal appearance characteristics. If an electrode unexpectedly shifts during the examination, an anomaly should rightfully be identified immediately when it occurs but the model should adjust to the new patterns to avoid massive false positive generation. If an anomaly detection algorithm does not possess online learning properties, there is no possible way to adapt to shifting distributions in the signal. It is therefore imperative for streaming anomaly detection algorithms to learn continuously in these real-world environments.

2.1.2 Time-Series vs Static Anomaly Detection

Anomaly detection on time-series data is also sometimes known as sequential anomaly detection. In any domain where data streams arrive in real-time such as in healthcare, traffic management of nearly any kind, hardware monitoring, intrusion detection and many more; data is collected typically in the form of time-series with relatively equal distance

collection intervals [21]. On the contrary, anomaly detection on data without any temporal information is known as static anomaly detection [104]. Static anomaly detection is only tasked with detecting spatial anomalies and is thus considered an easier problem that doesn't need as complex of models as time-series anomaly detection. Note that time-series data can include both spatial and more complex temporal anomalies. Algorithms that are fit for time-series anomaly detection must have the additional functionality of sequence memory and recognition beyond static outlier detection capabilities. In general, note that time-series data need not be ordered strictly by time. Time-series anomaly detection can be performed on data that has no inherent temporal ordering. For example, consider a signal that is sequentially recording the gender of people who are standing in a line. The inherent ordering of the data comes from the spatial position of each person in line. However, we can feed in the data sequentially in time to perform time-series anomaly detection the same way.

2.1.3 Classes of Time-Series Anomaly Detection

Many methodologies have been employed and tested to approach the problem of time-series anomaly detection including but not limited to lightweight statistical measurements, clustering based methods, various machine learning algorithms, Bayesian models, stochastic processes, deep learning and, recently, HTM [12, 59, 64, 130]. Different algorithms impose different demands on data. Some of these demands may be too strict and may become unrealistic for some real-world applications. Examples of these assumptions and demands on the data are the need for data labels and look-ahead to tag past anomalies. In many domains such as health and medicine the real goal is prevention rather than detection [64]. In these scenarios, it is valuable to detect anomalies as early as possible ideally conveying useful information well before a catastrophic failure. Other domains such as social media have underlying data distributions that change dramatically and frequently which then render past models that do not learn continuously obsolete. Creating new labeled datasets to keep up with changing data distributions is often impractical.

In this thesis, three mutually exclusive classes of time-series anomaly detection techniques that exist today are defined: batch processing, pseudo online and streaming. Several brief examples of modern algorithms that fall into these categories can be found in 3.2. These classes categorize the way in which an algorithm imposes demands and assumptions on the incoming data's structure, availability and temporal characteristics. Accordingly, we argue the class in which an algorithm falls dictates its applicability to different kinds of problems.

2.1.3.1 Batch Processing

The first class of time-series anomaly detection algorithms is known as batch processing algorithms. As the name implies, these algorithms process the data in batches and they are completely offline. It may very well be that any knowledge transfer between batches doesn't make sense for the given algorithm. For the purposes of anomaly detection, these batches usually need to be relatively large to capture an accurate depiction of normalcy in the data and recognize anomalousness. The size of batches for any realistic batch processing algorithm need to be far beyond just a handful of samples.

These algorithms require the entire dataset up front which is an unrealistic demand for any real-time applications. Especially in extremely time sensitive scenarios such as monitoring a patient's vital signs for unusual perturbations, action is often desired to be taken immediately in the presence of a significant anomaly. For many real-time applications, it is not practical to discover anomalies after a significant amount of time has passed as is the general working principle of batch processing algorithms when utilized on temporal data. Also unrealistic is the need of batch processing algorithms to store entire datasets at once. Streaming data accumulates quickly in general and can easily reach unwieldy sizes in modern applications.

These methods do have scenarios in which they work well. Retrospective analysis on historical data for example is well suited for batch processing algorithms. This type of analysis is often useful in domains such as analyzing past weather patterns or historical consumer behavior data in a business setting.

2.1.3.2 Pseudo Online

Some algorithms that are built for time-series anomaly detection can be considered to operate in a partially online or pseudo online fashion. What defines this class of algorithms is the requirement of an initial phase of offline learning on historical data or the need for look-ahead to future data to tag past anomalies in previously seen data. An offline learning process often needs to be repeated to accommodate for changing data distributions in these algorithms. This violates the underlying principles of a true online algorithm and this is not considered totally online. Only with enough processing power and complex parallel training schedules could a system that is partially online seem to be working in real-time. The need to store often massive training datasets is also cumbersome for real-time applications that are constantly receiving a stream of new data.

Most supervised strategies to anomaly detection would fall into this category. This is because an initial batch of training data and an associated training process is required

to learn a model prior to being applied to the actual streaming data. For example, artificial neural networks and deep learning fall into this category because of their (often expensive) offline training processes. Algorithms such as these would need to repeat their training strategy with newer data every so often in streaming scenarios because the underlying data distributions often change. For example, if the streaming application was monitoring a CPU's usage statistics in real-time, software updates and long-term pattern job frequencies would naturally change the underlying data distribution. These changes in the data distribution would result in consistent anomalies being reported until a new model was built. This phenomenon is shown in [64]. It's worth noting that it is also not always clear when to re-execute training procedures in these scenarios.

2.1.3.3 Streaming

Algorithms that qualify as streaming anomaly detection algorithms must meet several criteria to be applicable to real-time, streaming problems. The following six criteria are presented in [12] that identify an ideal real-world anomaly detection algorithm fit for streaming detection:

1. Predictions must be made online; i.e., the algorithm must identify state x_t as normal or anomalous before receiving the subsequent x_{t+1}.
2. The algorithm must learn continuously without a requirement to store the entire stream.
3. The algorithm must run in an unsupervised, automated fashion without any data labels or manual parameter tweaking.
4. Algorithms must adapt to dynamic environments and concept drift, as the underlying statistics of the data stream is often non-stationary.
5. Algorithms should identify anomalies as early as possible.
6. Algorithms should minimize false positives and false negatives.

An algorithm that possesses these properties can rightfully be called a streaming anomaly detection algorithm. In a sense, these criteria echo the requirements we would expect for a human if we were to hire a human to watch for anomalies on a real-time data stream. We wouldn't expect the human to need future data to recognize an anomaly nor would we expect them to need to store the entire dataset. We wouldn't expect to need to manually tweak their abilities, either. Learning for humans in the context of anomaly

detection is done completely autonomously. Human intelligence is flexible and efficient and thus is fit well for this task.

2.2 Hierarchical Temporal Memory Theory

Hierarchical Temporal Memory (HTM) is a biologically constrained theory of machine intelligence. HTM development has had a strong reliance on neuroscience to facilitate the construction of biologically plausible models that aim to capture the working principles of the human brain. In the long term, this is in the effort to exhibit strong intelligence in machines in the same way as we understand and observe it to be so in humans. In the short term, part of what HTM has brought to the field today is a theory of how the neocortex is capable of time-series sequence learning and prediction. An associated software implementation inspired by HTM core components from Numenta exists called NuPIC [120]. At the time of this thesis, the latest version of NuPIC is v1.0.3. NuPIC was the HTM implementation that was used in [12] and [64], so we use it in this thesis as the underlying implementation of HTM with the same hyperparameter settings for comparison purposes. More about NuPIC can be found in 3.1.

Concretely, HTM is concerned with the physiology and interaction of pyramidal neurons in the neocortex of the primate brain. In contrast to other neuroscience-inspired algorithms such as artificial neural networks [101], HTM introduces a biologically plausible and in-depth structural model of pyramidal neurons along with theories of their functionality, connectivity and collective organization in the neocortex. HTM overall not only includes everything mentioned above but also a philosophical theory of intelligence in general. This theory of intelligence is described in more detail below in Section 2.2.1.

2.2.1 What Is Intelligence?

From more of a philosophical standpoint, HTM defines intelligence in a way that is contrasting to more traditional interpretations such as is described in the Turing Test. A direct result of its design, the Turing Test asserts that behavior is the ultimate indicator of intelligent behavior [84]. In other words, what we call the "Turing interpretation of intelligence" suggests that it doesn't matter *how* a machine exhibits intelligent behavior. If it can perform that behavior, it is intelligent for all intents and purposes. In the context of the Turing Test, this means however a machine can trick a human into thinking it is a human communicator is irrelevant in the judgment of its intelligence. What is of chief importance is that it is simply able to do so. This driving force of achieving desired behavior by any means necessary has arguably shaped AI research since its early beginnings.

In contrast, HTM rejects the idea that behavior is the be-all and end-all indicator of intelligence. This is first motivated in [47] by the observation that humans can do intelligent things without any physical behavior at all. For example, humans can think and contemplate on subjects all within in their own minds. Humans do this without moving a single muscle or interacting with the outside world in any non-trivial way *and* without having a specific goal or desired behavior to emit. What currently separates man and machine in the way that they solve problems is the philosophical concepts of thought, understanding, creativity and sentience.

A famous thought experiment used as criticism of behavior-based intelligence is known as the "Chinese Room" experiment [24]. This idea was first introduced by John Searle in an article he authored called *Minds, Brains, and Programs*. It proposes the idea that something can perform a behavior without possessing any intelligence or understanding of what it is doing. The thought experiment is shown (in a paraphrased version) below:

Chinese Room Thought Experiment *You are bound to a closed room where Chinese characters are fed through a small window. You have no knowledge of Chinese either written or spoken. You only have a large book of instructions that enable you to correlate one set of symbols with another set. Your job is to execute these instructions blindly until you eventually output another set of Chinese characters. You then give the output characters back through the small window. Consider a sheet of Chinese sentences being fed through the window to you. You have no idea that these sentences are questions that possess concrete answers. You execute the instructions provided to you in the book by pen and paper. Your book of instructions is highly detailed and accurate so that the output characters you blindly generate are the correct answers to the original questions. When you feed those generated characters back through the slot, they are indistinguishable from any sentences written by a native speaker who also knew the answers to the questions. Anybody who reads your answers will have no idea that you do not speak any Chinese. From the point of view of an outside observer, the answers you gave are completely correct and not suspicious. However, as stated before, you do not actually understand Chinese and thus do not possess the capacity to form or convey novel ideas in the language. You only behaved like a computer; you performed deterministic computations on precisely and formally specified elements.*

The point of this experiment is to try to argue that just because something produces a desired behavior does not necessarily mean that it understands what it is doing and is thus not truly intelligent. There exist many criticisms of this experiment. One common criticism is that the person in isolation might not understand Chinese but the entire

system (the person, instructions and window) can be said to understand Chinese. In any case, there clearly exists a need to differentiate behavior and intelligence for clearer understanding.

2.2.1.1 The Essence of Intelligence

Instead of behavior, HTM proposes that accurate *prediction* is the true essence of intelligence and intelligent behavior alike. Prediction often precedes behavior and plays a key role in reflection afterward. Jeff Hawkins argues that the ability to predict the future shows true *understanding* and thus intelligence [47]. Hawkins illustrates in [47] that prediction is truly ubiquitous in human life as intelligent beings. Several neuroscientists and psychologists before Hawkins have suggested the importance of prediction as well such as Horace Barlow of the University of Cambridge who framed the argument as intelligence is all about making a guess that discovers some new underlying order. [17]

Humans are constantly predicting things everyday whether it be the answer to a question presented to them, what time you should leave for work in the morning or something as seemingly simple to us as how you should orient your muscles to take the next step as you are walking. Predictions are made on a much less tangible scale as well. We frequently don't even realize we are making predictions. We often unconsciously predict what an apple is going to smell, taste and feel like before we eat it which then form expectations for us. We do this for virtually every task executed in cortex often involving abstract ideas and combinations of sensory input. Combinations of senses is referring to the more abstract predictions that occur farther up in the cortical processing hierarchy. We often attribute lots of cross-sensory predictions associated with abstract concepts. For example, there's a good chance you can remember what your childhood home looked, felt and smelled like all the while recalling from your sensorimotor memory the sequence of muscle movements it took to get through the front door. HTM theory argues from observation that humans use predictions and the resulting expectations to make sense of the outside world. When predictions are broken, feelings of novelty and surprise are risen into our conscious awareness. The consistency and accuracy of our predictions are what gives us a sense of normalcy, consistency and understandability of everything that is happening around us and within us.

The connection between prediction and intelligence should be clear once one realizes how frequently humans engage in the act. The mark of an expert is to have a rich and deep understanding of a topic. This translates into the real world as the ability to make detailed, useful and accurate predictions of all kinds that serve to fill in missing information, solve problems, communicate things in a clear and precise way and to discover new knowledge.

At first it may not be clear how communicating clearly is related to prediction but when humans are communicating we are actively predicting what words and actions will convey our message the clearest. Ubiquity of prediction is true even within tasks that seem trivial to us like vision and standing on two feet. Our brains have been hyper-tuned by evolution over millions of years to solve these problems and thus in a sense we are all experts in them.

2.2.2 Biological Motivations

Numenta, as well as HTM in general, very openly use neuroscience as the ultimate source of inspiration and roadmap for development. Front and center on Numenta's website reads the following quote "Studying how the brain works helps us understand the principles of intelligence and build machines that work on the same principles. We believe that understanding how the neocortex works is the fastest path to machine intelligence, and creating intelligent machines is important for the continued success of humankind." [91]. However, it is not readily obvious if understanding and modeling the brain is the best path to take in the effort to build intelligent machines in the first place. Skeptics may offer arguments such as the fact that nature's solution to a problem isn't always the best from an engineering perspective. For example, we can engineer land vehicles that travel much faster than the fastest land animals and we engineer planes that fly much faster and higher than birds. Many may rightfully ask the question "why should we use neuroscience as a guide to develop artificial intelligence and, if so, to what level of detail is appropriate?" There are many differing opinions on this and there is not necessarily one correct answer. We will offer possible arguments to these questions here.

Why Should We Care About Neuroscience? Motivating the usage of neuroscience in AI research can be approached from two different perspectives. The first perspective is the most obvious one. One cannot deny the incredible and still mysterious capabilities of the human brain that modern AI currently cannot explain. It is worth noting that some applications in AI that are not explicitly inspired by biology can outperform humans in very targeted tasks such as IBM's Watson famously defeating the champions in the trivia style game show *Jeopardy!* [5, 35, 36]. However, the true strength of animal brains does not lie in its ability to perform individual targeted tasks (although they often do so incredibly well, too, albeit at the same level or secondary to machines in a very small handful of tasks). The first aspect of biological intelligence capabilities that modern AI cannot yet match is flexibility. The champions who were defeated by Watson were bested by a machine at the game of *Jeopardy!*, but they can do a virtually infinite number of other

mental tasks that Watson cannot without significant architectural changes and retraining procedures. *Human brains are general purpose machines in the highest degree.* One could imagine augmenting Watson's architecture to include the ability to discover the shortest path in connected graphs, but it would require vastly different algorithms augmented into the machine. Each new capability wished upon Watson would require these massive changes while the human brain can do virtually all these things with one convenient design. The second aspect is the capacity for more sophisticated mental tasks. There are several high-order mental activities that we humans perform every day that modern AI cannot explain. These kinds of activities chiefly involve things like creativity and truly robust and flexible reasoning strategies. In general, modern AI lacks the ability to create novel ideas and solutions as well as reason about ideas in an intelligent way. There have been many attempts in history to create AI capable of reasoning through the usage of rule-based logic systems, but these often entail impracticably massive databases of explicitly defined rules and facts that require frequent unmanageable upkeep. This methodology is incredibly inefficient and generally entails gigantic search spaces. The brain, in contrast, does not need to search and cycle through all its knowledge every time it wants to do something. When we see our mother's face, we do not search and cycle through every person we've ever met to conclude that that is our mother. Projects with these kinds of explicitly defined knowledge modeling techniques such as the OpenCYC project [3] which originally set out to encode all human common knowledge have seen considerable struggles. Famed AI researcher and Professor at University of Washington Pedro Domingos has deemed the project a "catastrophic failure" [32].

The second perspective is that of efficiency. Modern computers excel at storage and speed arguably because the only limit on how large they can be scaled up to is resource availability. Yet, it is brains that maintain the efficiency lead. One of the worlds most powerful supercomputers, called "The K" from Fujitsu is rated at 8.2 billion megaflops while the human brain has been estimated to operate at 2.2 billion megaflops. However, the K consumes approximately 9.9 million watts which to put in perspective is enough electricity to power 10,000 homes. The human brain on the other hand is estimated to consume only 20 watts which is less power than needed to power a dim lightbulb [39]. Doing the simple math, this means the K consumes 0.0012 watts per megaflop while the brain consumes only 9.09×10^{-9} watts per megaflop. With these calculations, the brain is operating approximately 132,805 times more efficiently. Human brains also fit nicely inside a human skull while the K is housed in four multi-story buildings totaling $21\,900\,\mathrm{m}^2$ of floor area [4]. Aside from the viewpoint of efficiently performing general computation, even specially designed modern AI systems suffer a similar fate in terms of efficiency. In

2011, when playing *Jeopardy!*, Watson required approximately 85,000 watts to operate and several rooms full of servers and cooling mechanisms [42]. From an algorithmic standpoint, essentially all modern machine learning techniques suffer from expensive offline gradient-based training procedures which especially do not scale well to real-time problems such as often seen in robotics applications.

How Detailed Should We Get? The entire premise of HTM is that the human brain is the best example of an intelligent system we have and it provides a road-map from which to follow to build intelligent machines. However, it is generally accepted among HTM theorists that creating an exact replica of the brain down to the lowest possible details is not a productive goal. Doing this very low-level modeling is sometimes known as whole brain emulation. One notable project using that working principle of whole brain emulation is the Human Brain Project (HBP) [79] founded by European Neurologist Henry Markram. The HBP aims for whole brain emulation for furthering our understanding of the brain and driving brain-inspired AI research. The HBP's approach was that of "bottom up" simulation in which as much low-level detail was collected as possible and then all of it was plugged into a supercomputer to see what comes out of the simulation. These methods have been highly criticized by many scientists who describe it as premature and fundamentally flawed [2]. Many believe it is premature to try to exactly recreate an organ which is still mostly a black box to experts. In the words of Mark Changizi, a neurobiologist at Rensselaer Polytechnic Institute in Troy, N.Y., "Lay people tend to think that we neuroscientists know what we're doing and that we're on the cusp of understanding it all, but that's so far from the truth" [49]. Projects like the HBP are typically focused on other goals such as increasing various brain disease understanding to facilitate the development of better treatments. But, there are inherent flaws especially with respect to tractability and efficiency in using this kind of methodology to drive artificial intelligence research. There are a few other similar projects around the world which face similar hurdles such as the China Brain Project [82] and the U.S.'s multi-billion-dollar BRAIN initiative [1].

As opposed to whole brain emulation, HTM theorists and engineers have taken a less strict approach to developing intelligent machines. It is the goal of those scientists to capture the functional principles of the brain which give it it's capabilities and implement them inside of an algorithm in a biologically constrained manner. Brains exhibit various structural differences between species and even within species on smaller scales. The most readily noticeable of which is size. The adult sperm whale brain is 8,000 cubic centimeters and the human brain is only about 1,300 cubic centimeters [81]. Yet, humans

exhibit mental capabilities that far surpass that of a sperm whale. Additionally, base intelligence necessary for survival like bodily control and the ability to search for food is found throughout the animal kingdom despite many different brain structural forms. This implies that the answer to intelligence cannot lie in the lowest level details.

Furthermore, as mentioned before, HTM theory is chiefly focused on understanding the human neocortex as opposed to other brain structures. The neocortex is the part of the mammalian brain believed to be involved in intelligent, higher-order brain functions such as sensory perception, cognition, generation of motor commands [70], spatial reasoning and language [73]. The neocortex is not only the newest part of the cerebral cortex to evolve, but, unique in humans, composes 90% of the cerebral cortex and 76% of the entire human brain [88]. It is widely known in evolutionary neuroscience that more primitive parts of the brain sometimes referred to the "hindbrain" function to support vital bodily processes such as heartbeat and respiration [93]. To specifically target the neural mechanisms which result in higher-order cognitive function, which AI scientists are almost exclusively interested in, the neocortex was chosen as a promising starting point. It is the opinion of Jeff Hawkins and many other neuroscientists that more than just the neocortex is very likely used in humans to achieve our level of mental capability [47]. Notable structures missing in HTM that very probably serve important roles in intelligence is the hippocampus and the thalamus, for instance. However, due to intense complexity and shortcomings in the completeness of neuroscience literature, we are unable to tackle the brain problem totally holistically and reasonably expect tractability or success.

It is worth noting the opposite end of the biologically-constrained spectrum. This approach includes a family of algorithms that maintain only a small semblance to biology but has arguably become a movement all on its own. These kinds of thinking strategies are best represented by a currently popular area of research known as deep learning. Deep learning is an incredibly pervasive and currently very practical area of research with many sub-fields and areas to specialize in [65]. Currently, deep learning has been very successful and is considered state-of-the-art for a large variety of supervised and unsupervised tasks such as speech recognition [41], object detection, natural language processing, and many more [65]. These algorithms have the loosest basis on biology and continued development sometimes abandon biology altogether. The entire field of deep learning can be traced back to the original "artificial neural network" (ANN). The only similarity that ANNs have to animal brains is a gross simplification of pyramidal neurons connected in a one-way feedforward fashion. While the modeling in ANNs do have some semblance with our understanding of neurobiology, many of the later advancements that have brought us into the age of deep learning lack sufficient evidence that supports

biological plausibility anywhere in the brain. This doesn't necessarily matter to deep learning researchers as employing mathematical and engineering tricks have resulted in very successful, albeit rather rigid and inflexible, models. Despite that there have been a multitude of papers published on the topic of ANNs, the core link to biology has not changed significantly despite massive leaps and bounds in neuroscience. Characteristic of deep learning is massive search spaces, inflexible and expensive learning procedures and the need for massive amounts of data. Geoffrey Hinton, one of the founding fathers of deep learning, has famously noted that AI research needs to "start over" to push materially ahead [9]. He is chiefly referring to a backbone learning principle in deep learning called backpropagation which holds no biological plausibility. It is the opinion of Hinton and many other AI scientists that unsupervised learning will allow us to solve more sophisticated problems in AI in the future more autonomously. Also spoken by Hinton, "Max Planck said, 'Science progresses one funeral at a time.' The future depends on some graduate student who is deeply suspicious of everything I have said."

2.2.3 Core Principles

In this section I will detail some of the most important core principles of HTM. This is not meant to be an exhaustive list; only the principles that are relevant to this thesis are discussed. These principles drive theoretical development and physical implementation at a high level. At the time of writing this thesis, software HTM implementations do not include all these principles. Chief among them is the absence of hierarchical processing in current HTM systems. However, development in the HTM community continues to progress simultaneously with neuroscience all the while guided by these principles.

2.2.3.1 Common Cortical Structure & Algorithm

In 1978, Vernon Mountcastle observed groupings of anatomically distinct structures which compose the primate cortex. He called these structures cortical columns because of the columnar organization of cells in which they consist. In Mountcastle's own words, he proposed "all parts of the neocortex operate based on a common principle, with the cortical column being the unit of computation" [85]. Furthermore, it was observed that the cells in each cortical column are organized into several layers and clusters of columns are grouped together into larger functionally-related cortical regions. Mountcastle's original observation has given rise to much of HTM theory in its current form. It has been later observed that the cortical columns in the neocortex can be organized into six distinct layers each with stereotypical patterns of connectivity and cell type concentrations [117].

In the context of HTM, the cortical columns inside the neocortex are often referred to as minicolumns or simply columns. Each layer of the columns in the neocortex have its own hypothesized function and interaction with other layers and columns. Much of the specifics regarding the hypothesized function of individual layers are discussed in [47].

More important than physical similarity, the observed ubiquity of organization and structure in the primate cortex lends itself to the suggestion that each region of the cortex is performing a common cortical algorithm. This suggests there is a unique underlying functional algorithm within cortical columns and their interactions that provide for higher order mental activities. The common cortical algorithm must then be completely agnostic to the original source of its input. Whether the brain is processing input from light, sound, heat, sensorimotor sensations, etc., the algorithm that processes and makes sense of this information must be the same. From an evolutionary perspective, the idea of a common structure and algorithm poses a clear advantage. With such a system, the animal need only replicate the core design many times to develop new functional brain areas as opposed to evolving a brand new design. Additionally, this core design is naturally flexible and the system can naturally tolerate internal faults as cellular resources can simply adapt to any new environment.

Interestingly, a very famous experiment has shown evidence of a common cortical algorithm through means of surgical manipulation. The ability of cortical regions to perform various functions has been demonstrated by an experiment in which visual input was surgically rerouted to auditory cortex in neonatal ferrets. The mature animals were able to respond to visual stimuli after retinotopic maps and typical visual receptive fields developed within their auditory cortex [124]. This experiment illuminates the idea that cortical regions are robust and can make sense of any input source if it has been encoded into a similar representation of cell firing patterns.

As noted in [45], this idea of a common cortical algorithm can lead to some unintuitive ramifications. For instance, its difficult to imagine the sensations of vision and balance as taking the same representational form inside the brain. We often do not imagine vision, especially object recognition, as a temporal sequence pattern recognition inference task, either; and it certainly isn't taken to be so in popular computer vision algorithms. Ample biological evidence suggests this to be the case, however [34]. In some sense, the adoption of the idea of a common cortical algorithm becomes freeing albeit initially confusing and unintuitive. The idea that many specializations in AI such as vision, speech, robotic motion, natural language processing and more can be solved through one common structural design and family of algorithms offers an explanation to the incredible flexibility of biological intelligence which is totally unmatched in traditional AI approaches.

2.2.3.2 Sparse Distributed Representations

Throughout natural life, the primate brain forever rests in a dark, quiet fluid-filled chamber just millimeters inward from the cranium. The only exposure to the outside world for the brain are the electrical signals on its input fibers. These signals come from every kind of sensory and feedback mechanism in the body whether it be the eyes, ears, spinal cord or any other structure that sends information to the central nervous system. The brain does not directly receive photons or vibrations from the outside world; the sensory signals must be transformed into the same representation space before they enter the brain for processing. This common representation space that the mammalian brain uses nearly ubiquitously is called sparse distributed representations (SDR) in HTM theory.

Sparse refers to the fact that only a few out of many thousands of neurons in a region of cortex are active at any given moment in time. Sparse neuronal activity has been observed in a wide variety of mammalian brain regions [37, 38, 108]. *Distributed* refers to the fact that the neuronal activation patterns are spread across an entire population of neurons. SDRs are not moved around in memory like data in a computer. It is not an explicit knowledge representation mechanism. Instead, sensory experiences as well as knowledge takes the form sequences of sparse distributed global patterns of neuronal activity.

It is proposed in HTM, inspired by observational results in neuroscience, that SDRs are the common language that the brain uses to encode and process information of all kinds [45]. Concretely, an SDR in HTM is a binary vector where each bit represents a neuron (or column). A bit in the on position indicates that the neuron is currently active and vice versa. It is common to use a sparsity level of around 2% in HTM systems; so only about 2% of neurons are active at any given time. Each bit inside of an SDR does not play a human interpretable role in general. Only when considering the activity of the entire population is semantic meaning captured. If two input vectors activate the same bit in an SDR, it means the original data have some semantic similarity between them. The more overlap in activations that two SDRs share, the more semantically similar they are. Note that semantic similarity can be defined in any way you like in the encoding scheme. For example, when encoding time, you may wish to encode 1:00 PM to be more semantically similar to 1:30 PM than 9:00 PM because it is closer.

There are several advantages that using SDRs provide to HTM due to their mathematical properties. The first advantage is robustness to noise and variation. If you think about the way data is stored in a digital computer, it is explicit, rigid and fragile. In the ASCII chart [6], we know each unique sequence of bits represents an entirely different character without any kind of semantic overlap. If you were to flip a random bit in an ASCII encoding, it would change to something completely new and different. SDRs do not

operate like this. Each SDR, when representing some state, is composed of a few on bits in distributed across many off bits. With a reasonable resolution, size and sparsity level of the SDR, this means a single or a few bit changes do not destroy semantic meaning. There is a linear relationship between bit overlap and semantic similarity instead of an all-or-nothing scheme. If you imagine there is a predefined number of specific neurons in an active state needed to recognize a pattern where each neuron plays a semantic role, you could flip half of those neuron's activity and still see a 50% semantic similarity which may very well be significant depending on your application domain and network design. Flipping half of the bits can be interpreted as 50% of the bits being subjected to noise and variation.

Normally, an increased tolerance to noise would have the tradeoff of accidentally recognizing incorrect patterns. Sparsity, however, plays a key role in maintaining discernibility and reliability of correct pattern detection in the face of this increased tolerance to noise. By maintaining sparsity in a large population of cells, we decrease the probability that any one specific neuron is activated by accident or coincidence. There is a massive number of possible random sparse patterns of neuron activity on an SDR of reasonable size in general due to a combinatorial explosion. If the SDR size is increased by one bit, the number of possible combinations grows with factorial complexity. Take for example a population of only 2048 cells. Assume 2% sparsity which means 40 active cells define a pattern of neuron activity. There are $C(2048, 40) = 2.37 \times 10^{84}$ different possible combinations of 40 active cells in this cell population space. Additionally, the following example is given in [46] to calculate the probability of a false match on a dendritic segment. Assume a population of 200,000 cells, a sparsity level of 1% (2000 active cells at any given time) and a threshold of 10 activated synapses on a dendritic segment to recognize a pattern and cause an NMDA spike. The probability of these ten synapses getting activated for the wrong pattern is only 9.8×10^{-21}. Furthermore, forming more synapses than necessary to recognize a pattern increases the resilience to noise and variation while maintaining a low probability of false matches. If we form twice as many synapses than needed to recognize a pattern, that means 50% of the neurons can be flipped but the pattern is still recognized. In the previous example, introducing a 50% noise tolerance only increases the chances of a false match to 1.6×10^{-18}, however. For a more in-depth discussion on especially the mathematical properties of SDRs than is offered in this thesis, the reader is directed to [11].

2.2.3.3 Streaming Data & Sequence Memory

HTM systems are designed to work on a continuous temporal stream of data that is constantly changing like the way brains receive data from our senses. This feature is what allows HTM to be applicable to real time problems. This is vastly different than most artificial neural network techniques that are designed to be, often repeatedly, trained on massive static datasets.

Temporal memory, or memory of sequence transitions, is used to both predict and infer patterns of stimuli as well as generate motor behavior in HTM theory. Only the former use is employed for anomaly detection in this thesis. HTM theory postulates that the fundamental function of every excitatory neuron in the neocortex is to learn and predict sequence transitions. Everything that follows from HTM is based on this assumption. The changes that occur in the sensor signal could come from the subject being observed changing or from the sensor itself changing. An example of the former could be the person you're listening to changing the pitch and volume of their voice as they talk, and the latter could be moving your eyes to focus on different parts of an image. In that way, behavior and attention are intimately linked with sequence memory and prediction. In the HTM implementation and application of this thesis, there is no concept of behavior, yet the same underlying principle of sequence transition learning is still present.

2.2.3.4 Local & Online Learning

Hebbian Learning [23] forms the basis of learning mechanisms in HTM systems. Stated simply, Hebbian Learning is a phenomenon where repeated activity between neurons strengthens their connections and vice versa. With each new change in the data signal, synapses on every dendritic segment of every HTM neuron are constantly being updated. Learning in HTM systems is completely local; occurring independently at the level of each synapse. Synaptic plasticity in cortex being localized to dendritic segments that have been depolarized via synaptic input immediately followed by a back-action potential has been observed in literature [71] but remains to be a widely established phenomenon in neuroscience. In any case, this is to be contrasted with global training procedures seen in artificial neural networks which minimize a global loss function through a gradient-based training procedure. This difference between local and global learning is an important one for systems employed in the real-world that are expected to learn lifelong. Artificial neural networks, and any other model fitting methodology that employ global learning properties, make global updates to the model whenever learning something new is desired. *The observance of catastrophic forgetting of previously learned information is an*

obvious consequence when employing global learning strategies. On the contrary, learning in HTM models is completely local where learning new information only inadvertently and infrequently effects previously learned information. As is briefly formalized in Section 4.3.3.1, the representational capacity of HTM models even with modest parameters is astronomical which leaves plenty of room for learning many different things incrementally.

Furthermore, learning in HTM is implemented in an unsupervised manner through a cycle of prediction, observation and reward or punishment. There is no need for any labels on data samples at all. The temporal memory algorithm is constantly forming predictions of the state of the internal network in the very near future. If a column prediction turns out to be correct, the active synapses on the dendritic segment that caused the prediction are strengthened. Oppositely, if the prediction is incorrect, the synapse permanence is decreased. This simple mechanism of synapse permanence updates elicited by activation patterns and prediction constitute the entire learning properties of HTM sequence transition memory.

2.2.3.5 Hierarchy & Invariance

The world is a complex, dynamic place that is full of variance. Even the most mundane tasks, although they seem simple to us, require the simultaneous integration and processing of often multiple sources of complex and variable sensory data. The levels of intensity of photons reaching different parts of your retina as you read this right now are constantly changing. Likely, the exact pattern of light hitting your retina at this exact moment of time down to the detail of individual photons has never been seen before in the history of mankind much less by yourself. Beyond that, from one second to the next, your eyes make numerous tiny muscular adjustments known as saccades. This means the patterns of light are constantly changing let alone unique at the photonic level. This variance extends to all senses beyond vision. How can this sensory data be used at all? The animal brain has developed amazing capabilities to effectively work with incredibly high levels of variance providing for an internal representation of the world that feels constant and discernible.

Every animal with a neocortex has a hierarchy of cortical regions. The number of levels and regions in the hierarchy varies between species, however. Human brains possess a neocortex with many deep folds that significantly increases the surface area and potential for deep hierarchical processing. It is well known that hierarchical processing is essential to represent abstract and high-level knowledge from low-level perceptions. Higher levels in the cortical processing hierarchy tend toward more general and invariant knowledge representations. In human vision, it has been noted that neuron firing patterns become more stable across forms of visual variance at higher levels of visual processing [31]. For

example, similar neurons fire at the top level of IT in the human visual system whenever a human face is being viewed no matter who it is or how they are situated in the visual field.

Through the principle of a common cortical algorithm, we know each hierarchical level is performing the same set of memory and algorithmic functions. This includes a vast amount of communication between cells in the region, the regions above and the regions below in the hierarchy. Artificial neural networks in their basic form include only communication in the forward direction during processing time. Recurrent neural networks can in some sense be including lateral within-layer connections. Jeff Hawkins offers some theories of the functional purpose of all three communication directions between hierarchical layers as well as how they are made structurally possible in the neocortex in [47]. In its current form, HTM theory has endeavored to understand what goes on inside a single layer before premature hierarchies are implemented. The HTM software implementations used in this thesis do not incorporate any hierarchical processing.

Chapter 3

Related Work

In this section I will offer descriptions of some related work in the problem that is explored in this thesis. The most relevant related work is by far NuPIC which is the software implementation of HTM that is headed by Numenta [120]. Additionally, I will discuss some of the other approaches to anomaly detection that exist with emphasis on streaming anomaly detection algorithms. Many of the streaming anomaly detection algorithms I describe are comparatively evaluated in Chapter 6.

3.1 NuPIC

The Numenta Platform for Intelligent Computing (NuPIC) [120] is an open-source HTM software implementation that was originally created by Numenta in June 2013. Numenta's implementation of HTM has undergone many changes since its original conception. At the time of writing this thesis, the most current version of NuPIC is v1.0.3. NuPIC v1.0.3 is architected in a way that allows for experimental code to be developed in Python but more efficient implementations of the core HTM algorithms to be developed in C++ in the "NuPIC Core." There exists many open-source variants on HTM implementations with various algorithmic differences and in various languages such as Java and Clojure [50, 92]. However, the codebase that is freely available in NuPIC is the original HTM codebase that was used in [12] from which this thesis is primary compared with to demonstrate the benefit of the HOPB framework. Thus, NuPIC is the underlying implementation of HTM that is used in all of the experiments.

Sections 3.1.1 through 3.1.6 detail the algorithms and structures of the most current generation of HTM implementations. This is not to be confused with the older, previous work from Numenta.

3.1.1 High-Level Design

An "HTM model" as implemented in NuPIC and discussed in this thesis is composed of a collection of components each with a unique purpose. There are 3 core components to the HTM models used in this thesis: the encoder, the spatial pooler and the temporal memory (or sequence memory) region. These HTM models are specifically designed to work with data that contains a temporal component. A high-level illustration of HTM's core components and how they fit together in an HTM model is shown in Figure 3.1.

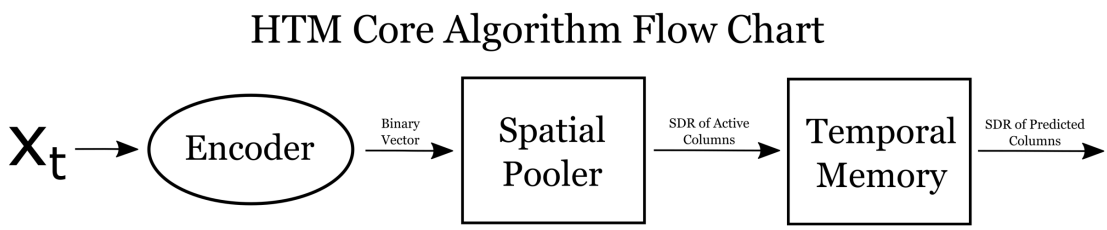

Figure 3.1: *Illustration of a full HTM model's core components.*

The data point at each timestep, x_t, is first converted into a binary vector representation that encodes the semantics of the data. The binary vector representation need not be sparse. From that binary vector representation, the spatial pooler converts it into a sparse distributed representation (SDR) that represents which columns in the temporal memory region have been activated by the feedforward input. The temporal memory algorithm uses those activated columns to preferentially activate the cells in each column that are currently in a depolarized "predicted" state, determine the next set of predicted cells for the next time step and lastly update all the synaptic weights to implement learning. Thus, the primary input to an HTM model is the timestep data and the primary output is the next timestep prediction of the state of active columns. The prediction of the next timestep state versus what got activated by the spatial pooler is what is used to perform prediction-based anomaly detection. There have been other HTM model variations built in NuPIC for other usages than anomaly detection, but they are not relevant to this thesis.

Furthermore, it's important to understand the terminology and organization of the model architecture. Within an HTM network, the most basic building block is an HTM neuron. Each neuron contains many dendritic segments and each dendritic segment contains many potential synapses to other cells in the network which represent lateral basal connections. These HTM neurons are organized into columns. In terms of the six-layer anatomy of the neocortex [56], the set of columns in an HTM network is made to model

layers two and three. Each cell in a column shares all feedforward connections to the spatial pooler output. That is, feedforward connections are only unique to columns. A visualization of the network at various levels of abstraction is shown in 3.2.

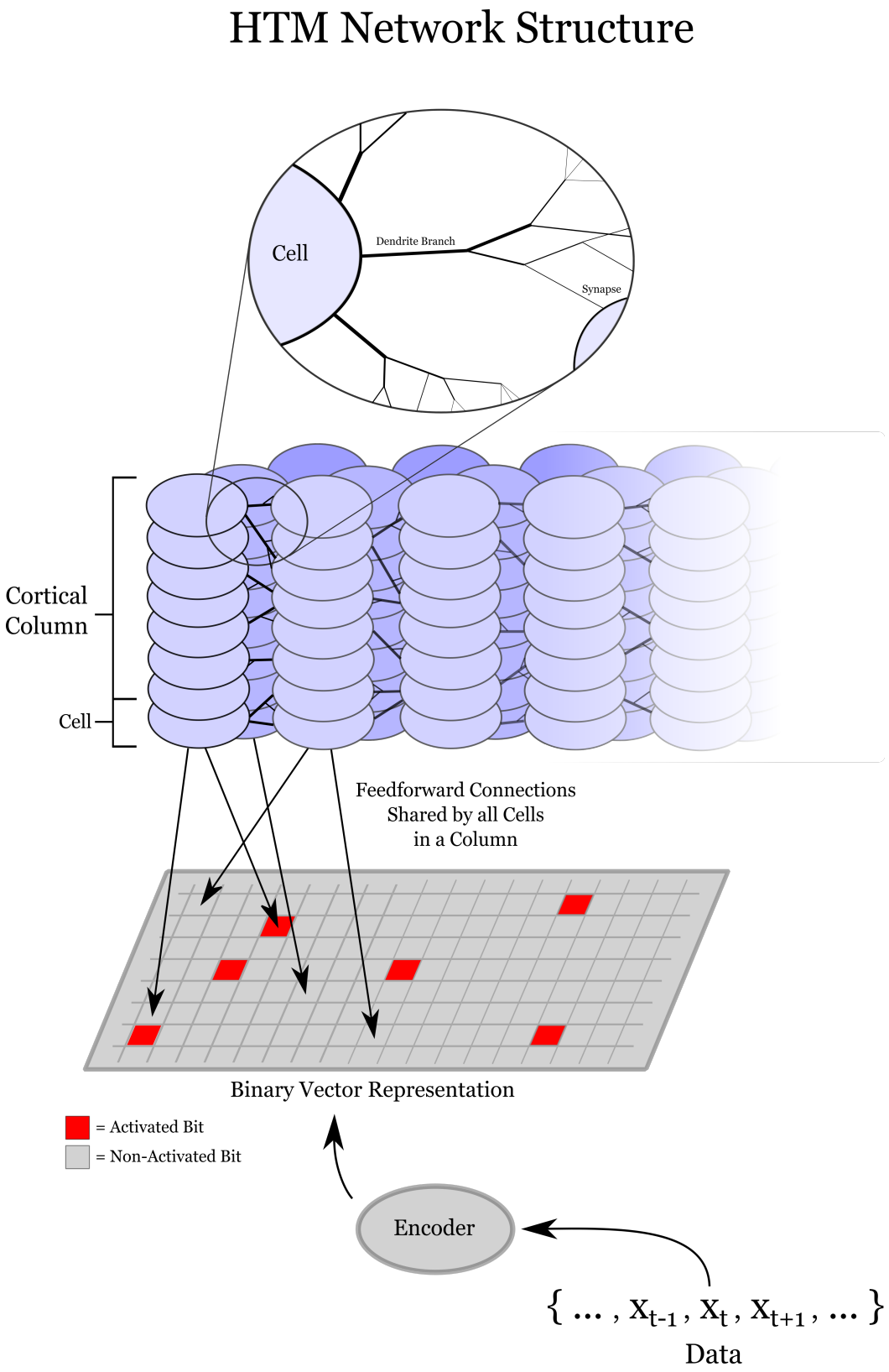

Figure 3.2: *Visualization of an HTM network at various levels of abstraction.*

Lastly, all synapses in HTM are modeled in the same way. Each potential synapse is associated with a permanence value that determines if a synapse has formed yet. If the synapse permanence is beyond a minimum permanence threshold, the synapse if formed and it contributes to NMDA spikes and action potentials. If the synapse is not formed, it does not have any effect. Note that synapses with a maximum permanence value and synapses with a permanence value at the minimum threshold have the same effect on spiking. This is okay because the network is built on massively distributed patterns of activity. Also, a synapse with maximum permanence has seen lots of reinforcement and can be interpreted as harder to forget since it would take many permanence decrements to erase the synapse compared to a synapse at minimal threshold permanence. This synapse permanence concept is visualized in Figure 3.3.

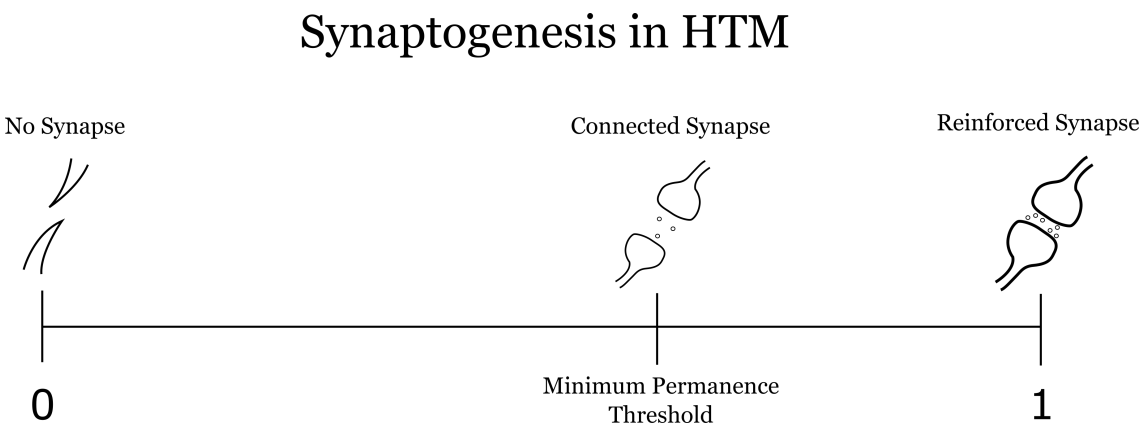

Figure 3.3: *Visualization of synapse formation in HTM.*

3.1.2 HTM Neuron

One of the foundational modeling details of NuPIC is the introduction of the HTM neuron. The HTM neuron can be considered an increase in modeling complexity and biological plausibility from the original perceptron model as originally introduced in [109]. Excluding numerous mathematical refinements, Rosenblatt's model is the neuron model seen throughout nearly all deep learning technology in some form. Rosenblatt's original perceptron model only goes so far as to consider the proximal synaptic input on pyramidal neurons. By this we mean the input to the neuron which can directly lead to action potentials; often called the feedforward input. A visual comparison of neuron model structure can be seen in Figure 3.4. Figure 3.4 comes from [46].

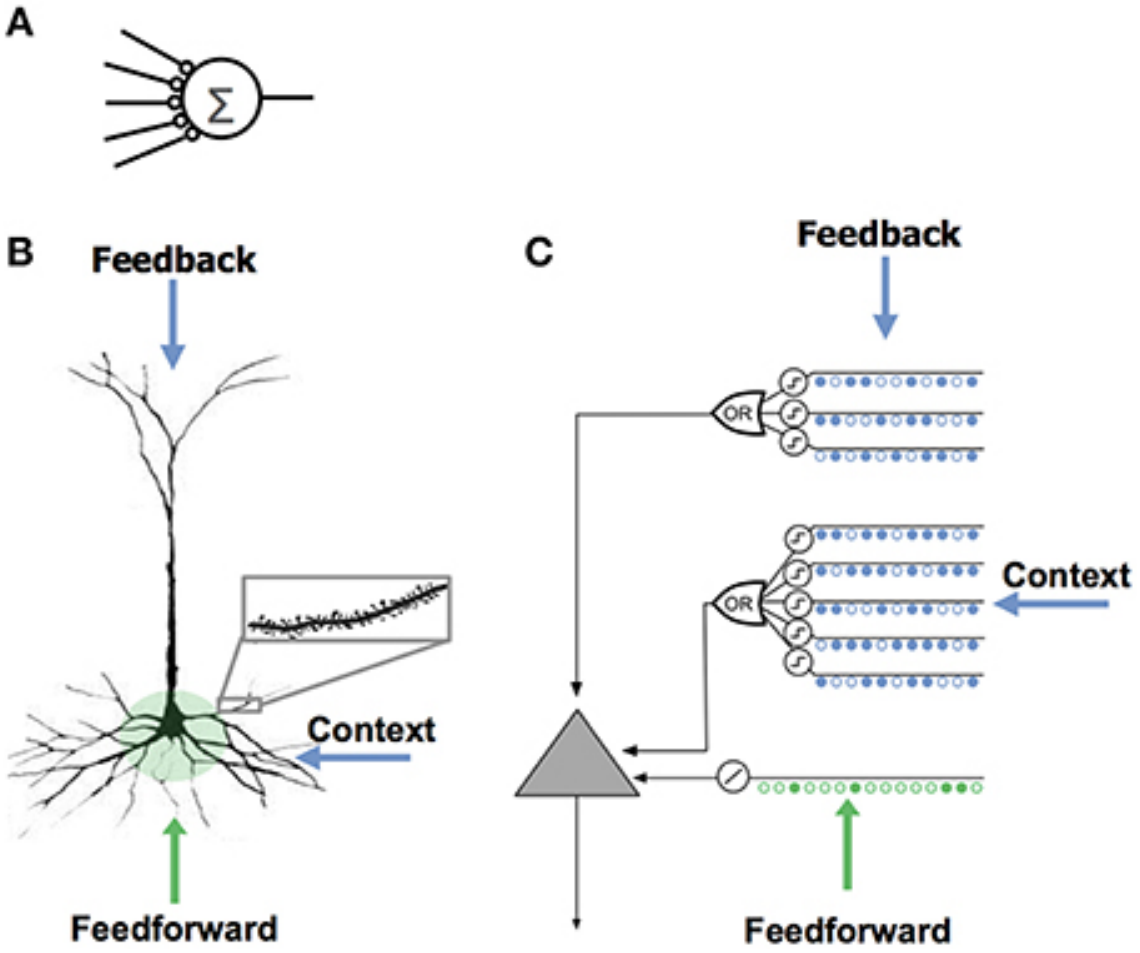

Figure 3.4: *A visualization of different neuron models for comparison purposes.* ***A.*** *A perceptron model featuring feedforward integration.* ***B.*** *A pyramidal neuron with three synaptic integration zones.* ***C.*** *An HTM neuron with three synaptic integration zones [46].*

The HTM neuron model incorporates signals from three different input zones. These three zones are the proximal zone, the basal zone and the apical zone. The proximal (or feedforward) zone consists of the dendrites nearest the cell soma which can directly lead to an action potential. The basal zone includes lateral basal dendritic segments which branch profusely as they stem outwards towards other cells and cortical regions. The inputs on these dendrites can be considered to provide contextual information from neighboring cells in the same cortical region [98, 105, 129] and are hypothesized to enable the neuron to learn transitions of network states. These dendrites do not typically lead directly to an action potential but instead bring the cell into a slightly depolarized "prediction" state through an NMDA spike also known as a "dendritic spike." [28, 76]. Experimental results show that simultaneous activation of synapses in close proximity on a basal dendritic branch

will combine in a non-linear fashion and cause an NMDA dendritic spike [63, 76, 112, 113]. It is believed the slightly depolarized cell is primed to fire early in this case and locally inhibit its neighbors in the process. This creates a competitive sequence of prediction and activation in which the cortex is biased towards activating its currently predicted state. The apical zone refers to the input coming from the dendrites which stem from the apex of the soma. As opposed to proximal dendrites, the distal apical tuft usually receives inputs from more distant cortical and thalamic locations. The effects on the cell body from apical dendritic spikes are largely unknown though there exist some theories on their true function. These dendrites have long been believed to relay feedback information from higher levels of cortical processing in the form of expectation or bias [60, 116]. The interaction between action potentials and proximal, basal and apical synaptic integration is still a hot topic of research [106]. More discussion of the HTM neuron and its properties can be seen in [46].

3.1.3 Sensory Encoders

As mentioned in Section 2.2.3.2, all different kinds of sensory information arriving on the input fibers of the brain must first be encoded into a common representation. The receptors on the retina along with the optic nerve in a human eye are a biological encoder: they convert external stimuli (light) into a form that is readable by the brain while encoding important semantics [95]. Semantics in terms of light hitting the retina is referring to things like where photons are colliding with the retina and with what frequency with respect to other areas of the retina. The cochlea is another example of a biological encoder which encodes the frequency and amplitude of sound waves into a sparse pattern of neuron activity using the stimulation of tiny hairs [126].

Encoding real-world streaming data into binary vector representations is an integral step of the HTM model. This is the job assigned to what are known as sensory encoders (or simply encoders) which represent the first step in the HTM algorithm. Sensory encoders play a role for HTM systems that is analogous to the role that biological encoders play for the brain. The input to an encoder is a timestep data value and the output is a binary vector. Note that the output of the encoder need not be in an SDR format; the output binary vector need not be sparse or distributed. The design considerations for sensory encoders are enumerated below.

1. It's very important that the encoder captures the semantic characteristics of the data (which are defined by the engineer) into the output binary vector. Semantics are captured by overlapping bits. This is to say semantically similar data should

result in binary vectors with a high number of overlapping active bits relative to non-semantically similar data.

2. In general, encoders must be deterministic in the sense that the same input should always produce the same output. Along those lines, the output of a sensory encoder always has the same dimensionality. The output binary vector is always the same size with activity distributed among all the bits.

3. Lastly, every possible binary vector that comes out of a sensory encoder should have a similar sparsity across the input space. This is necessary to work with the spatial pooler in its intended way. Enough density should be present, also, so that it is sufficiently robust to variation and subsampling for your application.

Consider, as a simple example, a binary vector output space of 6 bits.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Say you want to encode integers while defining the semantic similarity between them to be their closeness on the number line. One possible way to define the encoding is shown below.

$$1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \; 2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \; 3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \; 4 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$5 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \; 6 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Notice how the entire representation space has been used. In other words, the representation of each integer is evenly distributed across the entire space. This is desirable to ensure we are using the maximal representation capacity available to us. Using the entire output space, we were able to encode six integers total. However, notice that the above encoding fails to capture any of the desired semantic similarity in the data. Each integer has exactly no overlap with every other integer and thus no similarity is expressed between any integer. Alternatively, consider this possible encoding shown below.

$$1 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \; 2 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \; 3 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \; 4 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \; 5 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Now, only numbers that are next to each other on the number line have a single bit of

overlap with each other and thus some of the desired semantics have been encoded. But, notice we were only able to encode 5 integers using the entire representation space this time. We could go a step further as shown in the next possible encoding below.

$$1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \; 2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \; 3 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \; 4 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Now, notice how integers adjacent to each other on the number line will have two bits of overlap and integers that are within two integers of each other will have one bit of overlap. This means the underlying semantics have been encoded to a finer level of detail than the second possible representation. However, we were only able to represent four integers total.

This is a contrived example, but it illustrates the tradeoff between the level of semantic similarity that is encoded, otherwise known as resolution, and the breadth of possible states to be represented when the output size is fixed. There exist many possible ways to encode integers, decimals, etc. as well as non-numeric data types but this tradeoff is always one of many concerns. Encoding is rarely defined so explicitly per possible value. Instead, encoders are typically defined as functions on the input space. It is ultimately up to the engineer to make these decisions about encoding and it depends entirely upon the specific application's needs.

In theory, the underlying data, x_t, that is fed into the sensory encoder may be in any form with any number of attributes. Your temporal data may be in the form of scalar values, nominal classes, ordered values, text, images and anything else imaginable. The only requirement is that the data be encoded into a binary vector format that captures the important semantics of the data. In design consideration number one, it is mentioned that the engineer may define the semantics of the data however they please. This choice is entirely dependent on the underlying goal of the application. By designing the encoder, you are explicitly telling the HTM model what data are semantically similar and what aren't. There have been many proposed encoder designs as a result for even seemingly simple data formats like scalar values. A more in-depth discussion on encoders than what is offered in this thesis can be found in [103].

3.1.4 Spatial Pooler

Overview The job of the spatial pooler is to make sense of the messy, irregular input coming from the sensory encoders. In the brain, each cortical region receives converging feedforward input from multiple locations all with varying sizes and signal sources. Each region has no concept of where the given signal is coming from and what it's characteristics

will be such as the number of input axons the signal comprises. With multiple feedforward input sources all mixed together, it doesn't make sense that a region could uniquely process all these inputs without some common representation that it builds and operates on. The spatial pooler aims to create efficient normalized representations of all its input at a fixed size and sparsity while retaining the signal's semantic information so that the temporal memory algorithm can make sense of it. This is achieved chiefly through Hebbian-like learning principles, homeostatic excitability control, topological organization of sensory cortical connections and activity-dependent structural plasticity. All of these mechanisms are observed computational principles of the mammalian neocortex [23, 29, 57, 122, 132].

Concretely, the spatial pooler is named as such because the algorithm learns to group together spatially similar patterns into a common representation. Input patterns that are spatially similar are determined as such because they share many co-active bits in the encoder's output space. This has the effect of helping to learn invariant representations of abstract patterns where semantically similar patterns are naturally grouped together in the output representation. In an end-to-end HTM system (encoder, spatial pooler and temporal memory), the spatial pooler provides the SDR representation of the input that the temporal memory algorithm learns to predict. A single region in an HTM system is organized into a set of columns where each column has a set of associated neurons. The spatial pooler takes as input the output from the sensory encoders which are semantically encoded binary vectors. The output is an SDR of bits where each bit represents the activity state of columns in the HTM model. In that sense, the spatial pooler only operates at the level of entire columns in the cellular representation of the model. The spatial pooler decides which columns should become active and the temporal memory region decides from there which cells in the column to become active and which become depolarized. In the context of the three synaptic integration zones of a pyramidal neuron, the spatial pooler models the feedforward connections into the proximal dendrites. A visualization of the spatial pooler's structure and functionality is shown in Figure 3.5. Figure 3.5 comes from [27].

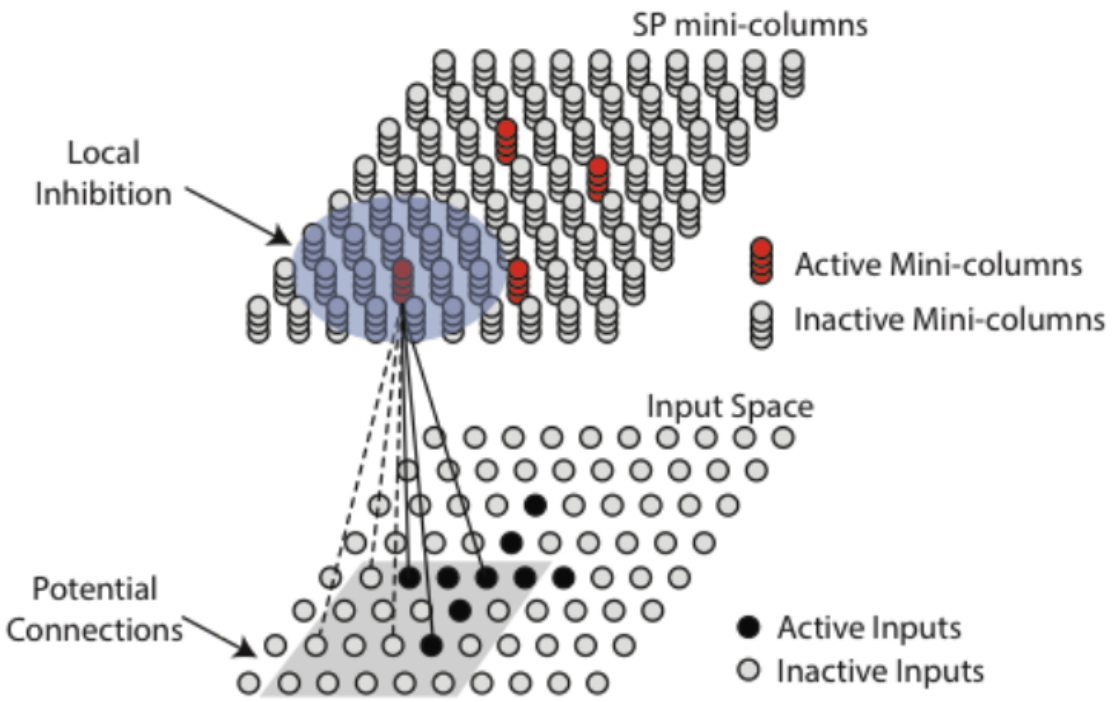

Figure 3.5: *A visualization of the spatial pooler's structure and functionality. Each column in the network is associated with a subset of potential connections to the input which is the output of the sensory encoder. Topological organization is encapsulated by the radius of the candidate area of potential synapse connections for each column. Local inhibition ensures only a small fraction of the columns which receive the most activation on its connections are activated within a given neighborhood. Synapse weights between columns and input cells are adjusted according to a competitive system of reinforcement and punishment based on activity levels. In addition, homeostatic excitatory control, called "boosting," increases the relative excitability potential of columns that are historically inactive for the purposes of ensuring utilization of the entire space [27].*

Algorithm Details The spatial pooler algorithm can be organized into three separate phases following an initialization.

1. **Initialization:** The spatial pooler must first be initialized before any input is received or any learning takes place. This is done only once. Each column has a candidate radius of input bits from which potential synapses may form called a potential pool. The initial list of potential synapses for each column is chosen randomly within its potential pool. For the purposes of this thesis, the number of columns is always chosen to be 2048. Since this is a relatively small number, the potential pool is the entire input space for each column. Additionally, each potential synapse that is chosen is given a random permanence value to start. The permanence value is

constrained to be between 0 (no synapse) and 1 (strongest possible synapse). These initial permanence values are chosen to be within a small range around the minimum permanence required to form a synapse. This ensures synapses can be connected and disconnected quickly upon the start of training as needed. The permanence values for each potential synapse for each column are also chosen to be higher towards the center of the candidate area on the input for that column to reflect the desired topological organization.

2. **Phase 1 - Compute Column Overlap Scores:** After the spatial pooler connections have been initialized, it is ready to take in input vectors. Recall these input vectors are the output of the sensory encoders thus they represent the signal source. The first step of processing these input vectors with the spatial pooler is to compute the overlap score for each column's synapses with that vector. The overlap score for each column is computed as the number of connected synapses (synapses that have a high enough permanence value) with active inputs multiplied by a boost value that attempts to mimic homeostatic control. We include a stimulus threshold at this point to ignore trace activations which are not significant. If the overlap score does not breach the stimulus threshold, it is set to 0.

3. **Phase 2 - Inhibition:** The next step in the spatial pooler algorithm is to select which columns should remain active after a process of local inhibition takes place. The inhibition radius is always proportional to the average receptive field size across all columns in the layer. The number of active columns allowable per inhibition area is chosen in advance as a parameter. If the number of active columns per inhibition area is chosen to be k, that means a column will stay active if it's overlap score is greater than the score of the k^{th} highest overlap score within its inhibition radius. This process of inhibition helps keep the sparsity of activation relatively constant. Inhibition is thought to help balance out excitatory firing in the mammalian cortex as they increase and decrease together. This balance is believed to be critical for proper cortical function [52].

4. **Phase 3 - Learning:** The last phase for a given input vector is the learning phase. The idea is to update the synapse permanence values for each potential synapse for each column based on a local competition-based learning mechanism inspired by Hebbian learning [23]. For columns that stay active through the inhibition process, for each of its potential synapses, if the synapse is active, its permanence value is incremented. Otherwise, if the synapse is inactive, the permanence value is decremented.

Boosting factors are also updated in this step. Recall boosting comes into play in phase 1. Updating the boosting factors involves measuring the time-averaged activation level for each column (known as an active duty cycle) and the time-averaged activity level of the column's neighbors. The boost factor is updated based on the difference between these two measurements. If a column has a low active duty cycle compared to its neighbors, it is incremented and vice versa. From an information theoretic perspective, it would be ideal for all columns to have a similar active duty cycle in every neighborhood. This boosting mechanism is inspired by numerous studies on homeostatic regulation of neuronal excitability [29]. Alternatively, if a column's time-averaged overlap score (known as an overlap duty cycle) is too low, its connected synapse's permanence values are increased.

Additionally, the inhibition radius (applying to all columns in a layer) is recomputed at the end of Phase 3 as the average receptive field size in the layer across all columns. The potential pool of synapses does not ever change for a column. However, the receptive field of a column only includes connected synapses (those with permanence values greater than the threshold). So, a column might have a large potential pool but a small receptive field if it has only formed synapses with the input connections nearest the center of its potential pool (those potential synapses near the center are initialized to higher values because they are "closer" topologically). These receptive fields change over time as the duty cycle and representational burden of columns fluctuates, so they must be recomputed to determine the extent of lateral inhibition between columns. In other words, there should be more inhibition going on for a region which is composed of columns with large receptive fields since in general that layer will have more active columns for its inputs. Increased inhibition in that scenario proportional to the average receptive field size helps ensure sparsity is stable.

Algorithm Properties There exist notable and desirable properties of the spatial pooler algorithm which are a direct result of its design. These properties help the temporal memory algorithm learn useful patterns in data in a continuous and flexible fashion. It is believed that the mammalian neocortex uses these same properties to facilitate learning in the real world. These properties are enumerated below.

1. **A Common Representation:** The spatial pooler forms representations of its input with a fixed size and sparsity. This is important because the input signals to a cortical region may arrive from many different sources and consist of varying numbers of cells that change over time. A common representation from which to

process on ensures useful synaptic adaptations can take place in temporal memory. In addition, fixed sparsity ensures each pattern has a similar probability of being detectable. If the sparsity of input activation could vary, input patterns with high activation density would be much easier to detect than input patterns with low activation density. This would result in a high false positive rate for high density patterns and a high false negative rate for low density patterns.

2. **Utilizing the Whole Space:** Through homeostatic excitability control, the spatial pooler is able to make use of the entire capacity of the network to create efficient and useful representations of the input. Neurons which fire disproportionately frequently or infrequently do not convey as much information as a balanced distribution of firing. In other words, a neuron that plays a role in many different patterns is accordingly less able to discern between its patterns and the information gained by its activation is less interesting. Conversely, a neuron that plays a role in very few or no patterns is going to be fired very infrequently and is going to play a small role in detecting patterns on the input in general. What NuPIC calls "boosting" (discussed above) is a method of ensuring the neuron firing patterns are efficiently distributed across the entire space of possible neurons. This is done by throttling the relative excitability potential of cells that are firing frequently and the converse. The biological counterpart of boosting is the observed modulation of synaptic efficacy and membrane excitability in neural activity in the brain that is thought to constrain neural plasticity and contribute to the stability of neural function over time [29].

3. **Robustness to Noise:** The act of pooling together semantically similar patterns into a single output representation adds a layer of noise robustness to the system. Real world data signals as well as patterns of cellular activity resulting from responses of sensory neurons can vary widely for a given stimulus. These small variations in the input can make the job of recognizing and making sense of them near impossible if not mitigated by invariant representations. Related to this, the spatial pooler is resilient to trivial patterns by maintaining synapse permanence values and minimum thresholds for activation. A "survival of the fittest" environment is in place when it comes to patterns that *are* and *are not* recognized. A potential synapse needs sufficient activation over time to drive its permanence value up and lead to a fully formed synapse. If the synapse is not fully formed, it does not contribute to any action potentials nor dendritic spikes. This prevents connections that are anomalous, erroneous or random from forming and influencing the system at all. A minimum threshold for action potentials and dendritic spikes prevent patterns from being

recognized unless sufficient confidence is in place conveyed by the presence of several connected active synapses beyond a minimum threshold. Without the threshold, many false positive activations would occur. This enables only the most clear and frequent patterns to dominate.

4. **Fault Tolerance:** Biological evidence for fault tolerance in the mammalian brain has been observed in patients with traumatic brain injuries such as a stroke. These patients experience damage to relatively large portions of their cortex. Typically sensory, cognitive and motor abilities may be diminished upon initial injury but are followed by substantial recovery [33, 89]. The spatial pooler's output representation is naturally fault tolerant due to the mathematical properties of SDRs (see the effects of subsampling in [11]) and the concept of self-adjusting receptive fields. By homeostatic excitability control, establishing minimum thresholds for activation, maintaining a large pool of potential synapses for each column and adjusting each synapse permanence based on activity level, the population of columns in an HTM layer will learn to best represent its input with it's given network parameters. If the column population size is reduced (or increased), the learning rules will dynamically adjust the output representation to fit the data with the new network characteristics.

5. **Online Learning:** Organic brains are highly "plastic." Regions of the neocortex can represent entirely different things in response to environmental changes. While fault tolerance refers to the spatial pooler's ability to tolerate internal faults, online learning is referring to the fact that it can dynamically adjust to changing patterns of the underlying data signal. The learning properties of the spatial pooler (at the level of individual synapses) are completely local and massively parallel. The adjustment of one synapse is independent of every other synapse. No offline training procedures are necessary as the data changes. As the distribution of patterns change in the data signal, so will the network in a completely autonomous fashion.

Further Reading For further discussion on the spatial pooler and experimental results, the reader is referred to [27]. For a mathematical formalization of the spatial pooler, the reader is referred to [83].

3.1.5 Temporal Memory

Overview The temporal memory algorithm simulates the hypothesized function of lateral basal dendritic connections emerging from the cell soma on pyramidal neurons in cortical regions. This kind of lateral synapse integration between neurons in the same

region is believed to learn and predict transitions in the network state of activation in the brain. This is analogous in function to the recurrent connections on recurrent neural networks and long-short term memory (LSTM) networks [55, 68]. The goal of the temporal memory algorithm in terms of the entire HTM model is to learn and predict which cells (and associated columns) will be activated next given the past activation states.

Each static state from the spatial pooler takes the form of a sparse distributed pattern of co-active columns in the network. However, further detail at the level of individual cells within columns provides additional representational capacity which includes past context. The same static input state which consists of a subset of active columns can take many different forms at the level of what individual cells in its associated columns are active. Each different possible state for a subset of columns represents a unique context in which we are observing that input.

The HTM network variables that are adjusted within the temporal memory algorithm include the predicted state and activation state for each cell in each column and the synapse permanences on each lateral basal dendritic segment on each cell on each column. The proximal synapses connecting columns to feedforward input from the spatial pooler are not used or adjusted in this algorithm. The fundamental input to the temporal memory algorithm is a subset of columns chosen to be active and the fundamental output is a prediction of cells (and associated columns) to be activated immediately next. The entire temporal memory algorithm can be explained as an initialization followed by a three-phase process, similar to the spatial pooler.

Algorithm Details

1. **Initialization:** Initializing the lateral basal connections must be done before any processing of input takes place. This need only be done once. Each dendritic segment on each cell is given a set of potential synapses to other cells in the layer with a random nonzero permanence value. These initial permanence values are chosen randomly with some synapses being above the minimum permanence threshold (forming a synapse) and some not.

2. **Phase 1 - Compute the Activation State:** Computing the new activation state of the cells of each column is the first step to take when a new input is received. Recall the input is a subset of columns that have been chosen to become active. If the column that a cell is contained in was not chosen and is not activated in the input, there is no chance that that cell is going to become active. If the containing column is chosen, there are two possible cases for the cells within the column to become

active. If there are no predicted cells anywhere in the chosen column, all cells within the column become activated. This is known as "bursting" and it stimulates new growth of synapses. Alternatively, if there exist some cells in a predicted state (from the previous iteration) then those cells and only those cells will become active.

Activating only the cells in a predicted state in an active column models the context in which a sequence state has been observed. In other words, an individual sequence state is represented as a subset of active columns but the individual cells active within those columns represent the context in which that sequence state is being observed. In this way, a single sequence state can take many different forms based on what came before it at the level of individual cells. In the example provided in [46], this allows the network to know to predict "D" and not "Y" if it knows "ABCD" and "XBCY" and it sees "ABC" as the input.

Note that even though the individual cells do model context when combined, that mechanism by itself is not capable of incorporating high-order context into the Numenta algorithm for anomaly detection. This is true for two reasons. The comparison of the predicted state to the representation of what "actually" happened, given by the spatial pooler, is at the level of entire columns which are effectively first-order always. Additionally, if a context shift occurs, the new data point is immediately adopted, and predictions are made as if that is the new and correct data point. If any previously learned connections from that data point exist, the previously learned sequence will be predicted to follow immediately after. In the case of a contextual anomaly, low prediction error will follow despite the anomalous context switch. The previous context is immediately forgotten. This behavior is illustrated in Section 6.2. A wider perspective of the HTM model's prediction for the sequence beyond independent first-order transitional predictions is necessary to recognize anomalous context.

3. **Phase 2 - Compute the Predicted State:** The second phase is to compute the new cells that should be put into a predicted state based on the new activation state. The lateral basal synapses on cells that have a large enough permanence will be activated if they are connected to an active cell. If enough activated synapses are simultaneously present on a dendritic branch, an NMDA spike occurs and the cell from which the dendritic branch originates will be slightly depolarized and put into a "predicted" state. This is controlled by a threshold hyperparameter representing the minimum number of activated synapses on a dendritic branch necessary to cause an NMDA spike. If the minimum threshold is not breached, the cell is not put into

a predicted state. Recall that these predicted states are predictions for the next timestep. Concretely, a cell in a predicted state means that the network predicts that the column in which the cell belongs will be activated in the next timestep.

4. **Phase 3 - Learning:** The last phase of the temporal learning algorithm is the synaptic adaptations that incorporate learning. This phase is like phase 3 in the spatial pooler algorithm with some additional details.

   The learning rule is still Hebbian-like in the sense that it rewards synapses that correctly predicted a column activation. There are two possible ways a dendritic branch can have its synapses reinforced. If a cell was correctly predicted (was previously depolarized and became active in phase 1), any dendritic branches off the cell that breached the minimum threshold and underwent an NMDA spike have its synapses reinforced. Secondly, if an activated column was unpredicted, there exists a need to choose a dendritic branch that will represent that pattern in the future. We simply choose the dendritic branch across all cells in the column that had the most synaptic input (even though it was below the threshold) to have its synapses reinforced.

   Reinforcing the synapses on the dendritic branches that were chosen in either of the two possible ways above is Hebbian-like. The goal is to reward synapses with active presynaptic cells (increase the permanence value) and punish those (decrease the permanence value) that don't.

   Lastly, if a cell was predicted but did not become active, we apply a small decay to the synapses on the dendritic branches of that cell that caused it to become depolarized. This punishes those synapses that lead to an entirely incorrect prediction.

Algorithm Properties There are some interesting properties of the temporal memory algorithm that will be enumerated and discussed here. These help the reader understand the design choices and the expected model behavior.

1. **Utilizing SDR Properties:** The mathematical properties of SDRs are used at nearly every step in the temporal memory algorithm and help it to function desirably. Subsampling of patterns is done on nearly every dendritic branch of each cell. In a population of cells, only approximately 2% of cells are going to be active at any given time due to local inhibition in the spatial pooler. Thus, a static pattern of activity is composed of roughly 2% of cells and some lateral dendritic branches on different cells might each end up learning to recognize this pattern. One might initially think the dendritic branch ought to be connected to all 2% of cells to recognize the pattern but,

the value of connecting to only a small fraction of those active cells to successfully recognize the pattern is practically the same. If the network is connected to only a tenth of those cells, for example, the probability that those cells are activated for a different pattern is extremely small if the pattern is large and sparse. This also allows the dendritic branches to connect to multiple patterns at once. There is a chance of false positive pattern detection if connections to multiple patterns exist on one dendritic branch; such as if partial activation on each pattern sum together to initiate an NMDA spike despite no single pattern appearing fully. If the network is large and sparse, the chance of this happening is minuscule. See Section 2.2.3.2 and [11] for the mathematics to back up these claims.

2. **The Effects of Multi-Cell Columns:** One of the many parameters in an HTM network is to choose how many cells should be allocated to each column in a layer. This can vary from one cell per column to arbitrarily many. However, choosing one cell per column is effectively like building a Markovian model that is only capable of first-order memory on each transition. Within first-order memory, the system only remembers what happened immediately before the current input and does not use any other past information. This obviously severely limits the capabilities of the network for any temporal problems with sufficiently complex patterns. Any input would always produce the same prediction regardless of what happened before. Each different pattern represents a different context of previously activated columns that came before it. Thus, the same input pattern can be modeled in many ways dependent on a variable amount of past context. The predictions that arise from a column activation pattern include a variable amount of past context as well. This is because if an active column contains any predicted cells, only those cells will become active. Accordingly, only the lateral synapses attached to those specific cells will become active which will significantly shape the prediction for the next timestep.

3. **Existing Limitations:** There exist some limitations with the current implementation of temporal memory that arise in practice. Firstly, it is known the number of different ways the same input can be represented in the network is very large even for modest sized networks. This means the representational capacity is large, but the downside is that the number of possible contexts is often so large that repeated occurrences of an input even in the same context of interest are often seen as brand new because of an arbitrary, ultimately unwanted, amount of past context coming into play. In other words, there doesn't exist any way to consolidate and generalize among different contexts to recognize the semantic similarity between inputs with

similar contexts and translate this into more intelligent prediction. The ability to generalize among contexts is called "temporal pooling." For an input in a given context to be recognized we know it had to have been seen before and lateral basal synapses would have had to form to lead those cells to be predicted. Each new context essentially demands a new set of lateral basal synapses which equates to a very massive number of these lateral synapses needing to be learned. The system will generally become more stable over time as synapses are forming but in practice this problem of fragility with respect to variable context often arises.

NuPIC has engineered several attempted solutions to this problem: resetting and backtracking. Resetting is the process of manually telling the network when a pattern has ended (and when a new one has begun). When a reset is called upon, the network state is reset and lateral connections from the end of one pattern to the beginning of another are forcibly not learned. This alleviates the above problem because when transitions are constantly learned between patterns, the amount of context being represented can run awry spanning multiple patterns in the past that may have no true bearing on the current data signal state. This resetting phenomena is clearly not autonomous, however, and is impossible for data without clearly defined patterns. Secondly, backtracking is implemented in NuPIC which is a way to manually search for previously known contexts in the event of high bursting activity. In other words, if an input comes along that is unknown (not well predicted) the algorithm manually searches for a context in which the input *was* known in the short-term past and locks onto that pattern (assumes we are in it) instead of building a new context. It is like manually trying to find when in the short-term past did a new pattern likely start since context in the representation has been lost. While this does sometimes help, it is limited by the fact that we must store an unintuitive and inflexible number of past network states to search through for a fitting context. It is in no way guaranteed that the best fitting context is going to be reachable in the short-term past at all. Additionally, it is an engineered solution without any known biological plausibility. It is the belief of the author that mechanisms of attention could alleviate the problem of autonomously identifying the beginning and ends of patterns and hierarchical processing could alleviate the issue of consolidating and generalizing among contexts.

Further Reading For further discussion and formalization on temporal memory and experimental results, the reader is referred to [46] and [26].

3.1.6 Previous Application to Anomaly Detection

The idea behind the Numenta anomaly detection algorithm presented in [12] is, in short, to use the predicted state compared to the actual state for each timestep as a measure of how anomalous the data in a timestep is, given previously learned patterns in the data signal. If a timestep is well predicted, it is assumed to not be anomalous and the converse. Getting a sense of how well a timestep was predicted comes down to measuring how many of the predicted columns became active. Concretely, we calculate the prediction error as in Equation 4.1.

$$s_t = 1 - \frac{\pi(x_{t-1}) \cdot a(x_t)}{|a(x_t)|} \tag{3.1}$$

Where $\pi(x_t)$ is the predicted state of columns and $a(x_t)$ is the activated state of columns. $|a(x_t)|$ is the number of on bits in the activation SDR. The dot operation, $\cdot$, measures the number of on bits in common between the two SDRs.

Thresholding the prediction error score alone would not work well because each dataset is going to have some inherent level of noise and error. This would result in many false positives in general. Instead, we model the most recent past prediction errors with a rolling normal distribution. With a window size of W, the sample mean μ_t and sample variance σ_t^2 are calculated as in Equations 4.3 and 4.4, respectively.

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} s_{t-i}}{W} \tag{3.2}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (s_{t-i} - \mu_t)^2}{W - 1} \tag{3.3}$$

The probability of the recent short-term average of prediction error is then computed and thresholded as the final determination of anomalousness. The probability is computed as the Gaussian tail probability [30] as in Equation 4.5.

$$L_t = 1 - Q\left(\frac{\hat{\mu}_t - \mu_t}{\sigma_t}\right) \tag{3.4}$$

Where $\hat{\mu}_t$ is the short-term sample mean calculated with a smaller window than W. If L_t is greater than a predetermined threshold, we declare an anomaly for the timestep.

3.1.6.1 First-Order Limitation

We discuss how the Numenta algorithm for anomaly detection as described in 3.1.6 suffers the pitfalls of using only first-order predictions here. The Numenta algorithm must decide

on the anomalousness of an instant of time, for every instant of time, independently. It is known that the temporal memory algorithm is learning next timestep state transitions as opposed to entire sequence representations [46]. Furthermore, it is shown in [46] that high-order multiple, simultaneous sequence memory with HTM networks is only achievable through columnar cell organization in which sequence states can take multiple representations within its associated column subset based on what came before it. If there were only one cell in every column, you are effectively left with a first-order Markovian predictor. Experiments in [46] show the strong limitations on performance that a first-order predictor imposes in the presence of high-order sequences. The issue is that the Numenta algorithm compares the internal temporal memory network representation only at the level of entire columns with strictly next timestep predictions to estimate prediction error for each timestep independently of the rest. As a result, the Numenta anomaly detection algorithm does not include any sense of high-order temporal sequence context in its computation of prediction error. Numenta's anomaly detection mechanism is illustrated in Figure 3.6.

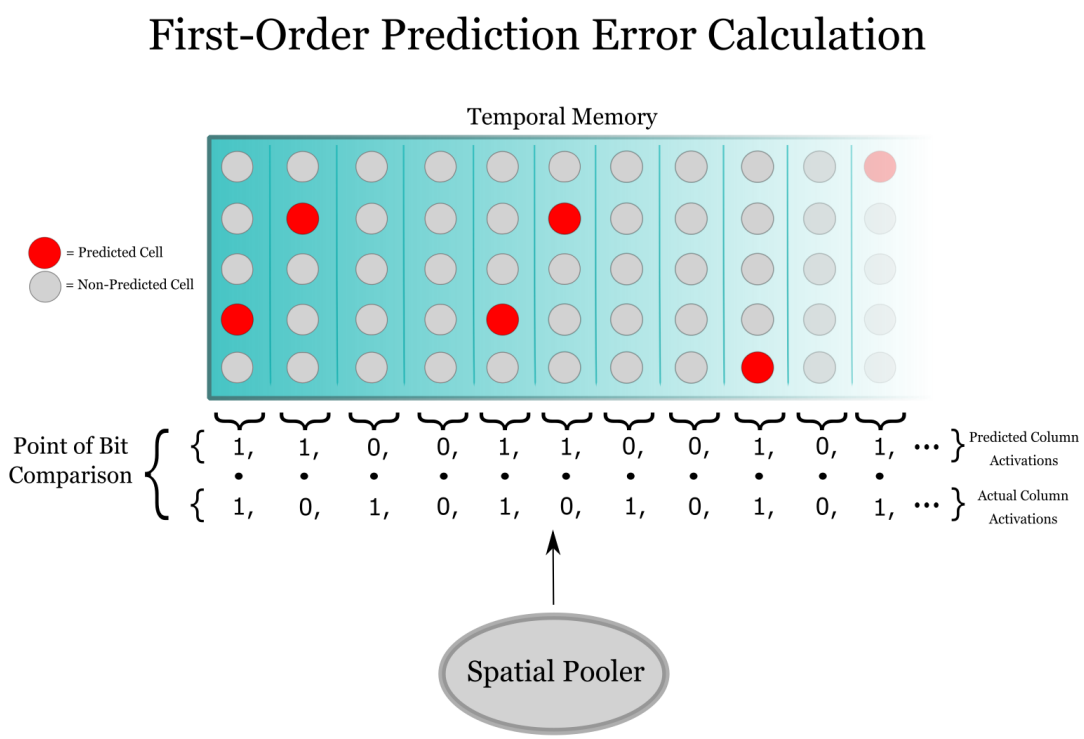

Figure 3.6: *Illustration of calculating prediction error in the Numenta algorithm for anomaly detection. The point of bit comparison is between the columns activated by the spatial pooler and the columns in the temporal memory region which either have or do not have a depolarized cell anywhere within them. This squashes the 2D representation of the temporal memory region into a 1D bit string which discards high-order sequence memory information.*

Comparison at the level of entire columns is necessary because the output of the spatial pooler is what is used as the "actual" activation state from which to compare to the "predicted" activation state given by the temporal memory algorithm. Excluding synaptic weight updates which transcend timesteps, the spatial pooler decides only which columns to activate based on each feedforward input in isolation; it has no concept or knowledge of high-order sequence memory or the individual cells within columns. When a context shift occurs in the data, the HTM network immediately forgets the context it was previously in and starts predicting off the newfound context. If the anomalous context shift cannot be detected in a single transition, it will generally go undetected. Additionally, clusters of high prediction error should prompt greater cause for concern, yet, in the Numenta algorithm, each first-order prediction is considered independent of

the rest. All these things combined, shown in Section 6.2, prevent the Numenta algorithm from incorporating local sequence context or the accumulation of broken expectations over time into prediction error which are both often integral for correctly recognizing anomalous state.

3.2 Other Algorithms

There is a plethora of existing strategies to the task of time-series anomaly detection. Many of these strategies target the same problem yet demand vastly different demands on data. There are three mutually exclusive classes of time-series anomaly detection that are enumerated in this thesis. A description of what defines each class is given in Section 2.1.3.

In this section, we will provide examples of modern algorithms that fall into each category. Several of the discussed streaming algorithms are included in the NAB repository [120] and are comparatively evaluated with HOPB in Section 6.7.

3.2.1 Batch Processing

There are numerous time-series anomaly detection algorithms which fall into the batch processing class. Netflix's Robust Anomaly Detection (RAD) [127] is one such example. This algorithm is based on the Robust Principle Components Analysis (RPCA) method [18] which identifies a low-rank representation of the data, noise and a set of outliers by repeatedly calculating the singular value decomposition of the data matrix and applying thresholds to the singular values and error for each iteration. This method is specifically designed for data with high cardinality such as the data obtained at Netflix. This algorithm qualifies as batch processing because the entire data matrix is needed up front to apply RPCA and search for outliers.

Another algorithm which requires analyzing the entire dataset up front is Yahoo's Extensible Generic Anomaly Detection System (EGADS) [62]. EGADS uses a large collection of anomaly detection and forecasting models. EGADS works by first building a time-series model of the data matrix which is used to compute the expected value at each timestep. Then, anomalies are identified by comparing this expected value with the actual value at each timestep. EGADS automatically determines the most likely anomalies through a probabilistic anomaly filtering layer. At a high level, EGADS can be thought of to work like HTM but there is currently no inclusion of self-tuning models which learn in a completely unsupervised, online manner. EGADS instead requires periodic batch generation.

Another example is called Heuristically Ordered Time series using Symbolic Aggregate ApproXimation (HOT SAX) [58]. This method reformulates the problem of time-series anomaly detection in a new light. The method is focused on discovering what they call "time-series discords" which are subsequences of a longer time-series that are maximally different to the rest of the time-series subsequences. This requires a rigorous definition of what it means to be maximally different as well as an intelligent search strategy to discover such subsequences. Perhaps implied by name, this method requires the entire dataset up front and is not suitable for real-time streaming applications.

3.2.2 Pseudo Online

Like batch processing, a large variety of time-series anomaly detection algorithms fall into the pseudo online class. Recall any algorithm or framework that requires an offline training process would fall into this category. For instance, [15] and [77] introduce the usage of Long Short Term Memory (LSTM) networks for the purposes of time-series anomaly detection. The procedure in [77] for using LSTMs for anomaly detection is strikingly similar to the one used by Numenta with one key difference. Firstly, an LSTM model is trained on sequence data that is considered normal behavior and a time-series prediction model is built. Then, like the Numenta algorithm, the LSTM is tasked with predicting each timestep based on what came before it. The measured error in the LSTMs prediction is used as a measure of the timestep's anomalousness. Also like the Numenta algorithm, these prediction errors are modeled as a multivariate Gaussian distribution to assess the likelihood of a true anomaly. The key difference between this work and the Numenta algorithm is that the LSTM must undergo its training procedure offline and does not learn continuously. For instance, if the characteristics of the underlying data distribution changes (sometimes referred to as concept drift), a brand-new LSTM would need to be trained and there is no automatic way provided in the work for detecting when and how this should occur. If retraining does not occur, massive numbers of false positives would ensue. HTM networks continuously adapt to new patterns and adjustments to the model are made simultaneously while making predictions. Also note that LSTM-based approaches judge prediction accuracy by how well the model was able to predict the actual timestep value while HTM-based prediction accuracy is judged by how well the model was able to predict its own internal state.

In addition, most clustering-based approaches fall into this class. One such example is called Online Novelty and Drift Detection Algorithm (OLINDDA) [54] that is based on the k-means clustering algorithm geared toward detecting concept drift and novel data points. Originally designed for network intrusion detection specifically, another example is

called self-adaptive and dynamic k-means [66] which combines a self-organizing map and k-means for modeling normal data stream behavior. This algorithm requires an initial offline learning period to learn weights prior to detecting any anomalies. Additionally, the Kernel Estimation-based Anomaly Detection (KEAD) algorithm is an anomaly detection method based on Kernel Density Estimates which requires look-ahead to future timesteps to decide the anomalousness of temporarily flagged timesteps. The KEAD detection threshold is also advised by the authors to be determined prior to actual use of the algorithm during an offline training period.

3.2.3 Streaming

One kernel-based approach originally designed for large datasets with high-dimensional features is called EXPected Similarity Estimation (EXPoSE) [114]. EXPoSE does obey the criteria of a true streaming anomaly detection algorithm. EXPoSE is capable of efficiently computing the similarity between new timestep values and a learned distribution of normalcy in an incremental fashion.

Another common approach to streaming anomaly detection is to use computationally lightweight statistical techniques. Examples of this include weighted moving averages [131], outlier tests [48], changepoint detection [10], hypotheses testing and exponential smoothing [118] and typicality and eccentricity analysis [25]. One such algorithm originally designed for anomaly detection in IP networks combines AutoRegressive Integrated Moving Average (ARIMA) and an improvement of the traditional Holt-Winters method [96]. Another popular framework in this category is called Skyline from Etsy [7]. All the algorithmic anomaly detection occurs in the analyzer service in the Skyline suite. Skyline relies on the consensus of multiple statistical algorithms. If most of the algorithms agree that a certain timestep value is anomalous, it will be labeled as such. In that way, it can be considered an ensemble technique. The basic algorithms included by default are simple techniques that are only meant to act as a starting point. Note that these lightweight statistical techniques tend to focus only spatial anomalies and fall short in their ability to detect complex temporal anomaly models.

Chapter 4

Proposed Framework

In this section, I will detail the proposed framework for High-Order Prior Belief (HOPB) predictions as well as how to apply it to streaming anomaly detection with HTM. Recall a popular theme to solve streaming anomaly detection is using predictions in time. This is to first build a time-series model of the stream (learn its normal patterns) then predict each timestep in real-time. A timesteps predictability is used to capture anomalousness. HOPB takes this high-level methodology and re-imagines it. The core idea is to extend independent, instantaneous predictions to overlapping chains of prediction. This has the effect of increasing fault tolerance and will better illuminate local context.

In this thesis, HOPB is formalized in terms of HTM as the underlying time-series model. We derive high-order predictions in HTM by establishing a connection from output to input in the temporal memory (TM) region of the HTM model. Repeating this cycle in a loop enables us to follow its predictions out in time. The output of the TM region that is referred to here is the predictive state of the network. The input to the TM region that is referred to here is the subset of columns that are selected to become active by the spatial pooler. Normally, the subset of activated columns come from the spatial pooler which forces a constant sparsity and reflects the incoming sequence state. In this case, the subset of activated columns is assumed to be exactly the columns that were predicted in the previous iteration. In this way, HOPB predictions can be thought of as "predictions on predictions."

Using these HOPB predictions for the effort of anomaly detection is to collect a set of predictive states for each timestep which take the form of sparse distributed binary vectors or sparse distributed representations (SDR). Each predicted state collected for a timestep was made at a different point in the past. In the Numenta algorithm, only the predicted state made immediately prior to a timestep is used. Synaptic growth and decay

on the basal dendritic segments which compose HTM's learning mechanism are turned off while capturing HOPB predictions for obvious reasons. There is no sense in learning on input that was predicted and might not be correct. Each collection of predicted states for each timestep contains contextual sequence information in this sense beyond first-order transitions. A comparison is made between each predicted state and the actual state which is captured by the spatial pooler in the real time stream where learning is constant. Each comparison gives us a sense of high-order prediction error made without the influence of new data and incorporates contextual information. After each comparison, we can combine each comparison in an intelligent way to obtain a sense of overall prediction error for the timestep immediately as it occurs which is then labeled as anomalous or normal.

We know that multiple possible futures can be predicted at once on an SDR. In that case, if the synapses are properly learned, branching futures will all be represented on the SDR together. The positive effect this has for our purposes is that no matter which of the possible futures proceed from the given sequence state, if the sequence is known by the network, it will have sufficient overlap with the predicted SDR. Only in the case where a future that is not known proceeds will sufficiently small overlap occur. This means we do not need to manually search for which future fits our predictions best. As long as the patterns remain large and sparse, collisions of patterns and accidental matching between patterns don't pose a threat.

4.1 Summary

When running in real-time, the way that predictions stack on top of one another is visualized in Figure 4.1.

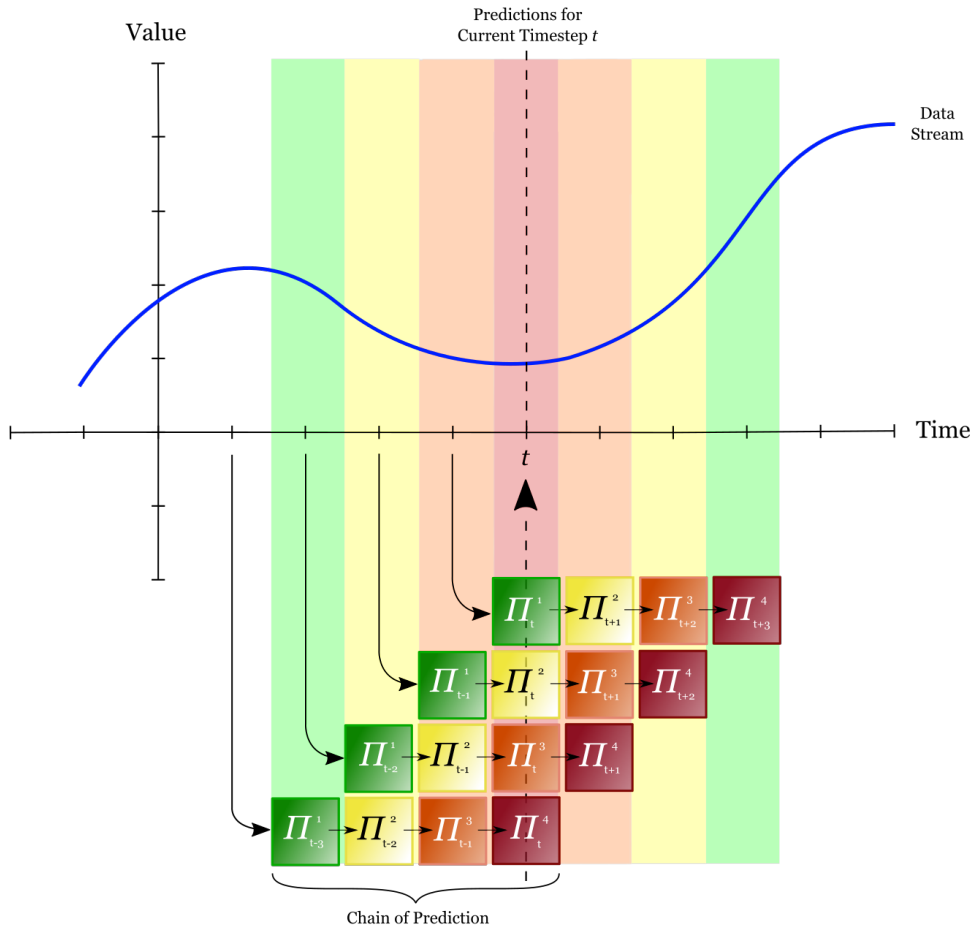

Figure 4.1: *Visualization of the stacking of predictions for timesteps when using HOPB. Slicing vertically through the predictions gives us multiple predictions for the timestep all made at different points in time.*

1. A new timestep value x_t is read and spatial pooler column activations $\mathcal{A}_t$ are generated.

2. A new chain of predictions $\vec{C}_t = \{\Pi^1_{t+1}, \Pi^2_{t+2}, \ldots, \Pi^{n_t}_{t+n_t}\}$ is obtained (see Section 4.2) whose length n_t is dynamically determined by quality and performance markers (see Section 4.3.2).

3. $\vec{C}_t$ is augmented into the HOPB manager data structure (see Section 4.4.1) and we return an accumulated list of predictions for that timestep, $\vec{P}_t = \{\Pi^1_t, \ldots, \Pi^i_t\}$ whose

contents are determined by the lengths of several previous chains. There is always guaranteed to be at least a first-order prediction for the current timestep but every other order of prediction may or may not be present.

4. Each prediction in $\vec{P}_t$ is compared with the current spatial pooler activations obtained in step 1 to generate prediction error scores $\vec{S}_t = \{s_t^i \,|\, \forall i \in \vec{P}_t\}$. Each s_t^i is an inverted bit overlap normalized by the size of $\mathcal{A}_t$ shown in Equation 4.1. It is the percentage of activated columns that were guessed *incorrectly*.

$$s_t^i = 1 - \frac{\Pi_t^i \cdot \mathcal{A}_t}{|\mathcal{A}_t|} \tag{4.1}$$

5. Each $s_t^i \in \vec{S}_t$ is arithmetically averaged together to get an overall measure of prediction error $\bar{s}_t$ for the current timestep t.

6. $\bar{s}_t$ is arithmetically averaged with the previous final score, f_{t-1}, to derive the final score for the current timestep, f_t, as in Equation 4.2. We call this the accumulating average because it is meant to detect and isolate accumulations of error which have more evidence of anomalousness.

$$f_t = \frac{\bar{s}_t + f_{t-1}}{2} \tag{4.2}$$

7. Similarly to Ahmad et al. [12] and Malhotra et al. [77], f_t is passed to an anomaly likelihood model which models a window of past final scores as a rolling normal distribution. This accounts for the normal level of unpredictability in the signal. With a window size of W, the sample mean μ_t and sample variance σ_t^2 are calculated as in Equations 4.3 and 4.4, respectively.

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} f_{t-i}}{W} \tag{4.3}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (f_{t-i} - \mu_t)^2}{W - 1} \tag{4.4}$$

The probability of a recent short-term average of the final scores $\hat{\mu}_t$ with respect to the rolling normal distribution of final scores is computed and thresholded as the determination of anomalousness. The probability of an anomaly p_t is computed as the Gaussian tail probability [30] as in Equation 4.5.

$$p_t = 1 - Q\left(\frac{\hat{\mu}_t - \mu_t}{\sigma_t}\right) \tag{4.5}$$

Where $\hat{\mu}_t$ is the arithmetic average of $\{f_t, f_{t-1}, \ldots, f_{t-w-1}\}$ where $w << W$.

8. A probability threshold τ is used to determine the anomalous tag. $p_t \geq \tau$ generates an anomalous tag while $p_t < \tau$ generates a non-anomalous tag.

4.2 Getting HOPB Predictions in HTM

To acquire high-order predictions of cellular activations, we need to follow the sequence of predicted states of the network out in time. Discussed further in Section 3.1.5, we see the lateral basal dendritic connections serve as independent pattern detectors to place the cell in a slightly depolarized state ready to inhibit its neighbors when the column is presented with feedforward activation. In other words, a column that contains one or more cells in a predicted state represents the fact that the HTM model thinks that that specific column is going to be activated in the next timestep. We can then assume the prediction was correct and activate that column to find the next columns that would get depolarized after it. The method of HOPB predictions is illustrated in Figure 4.2.

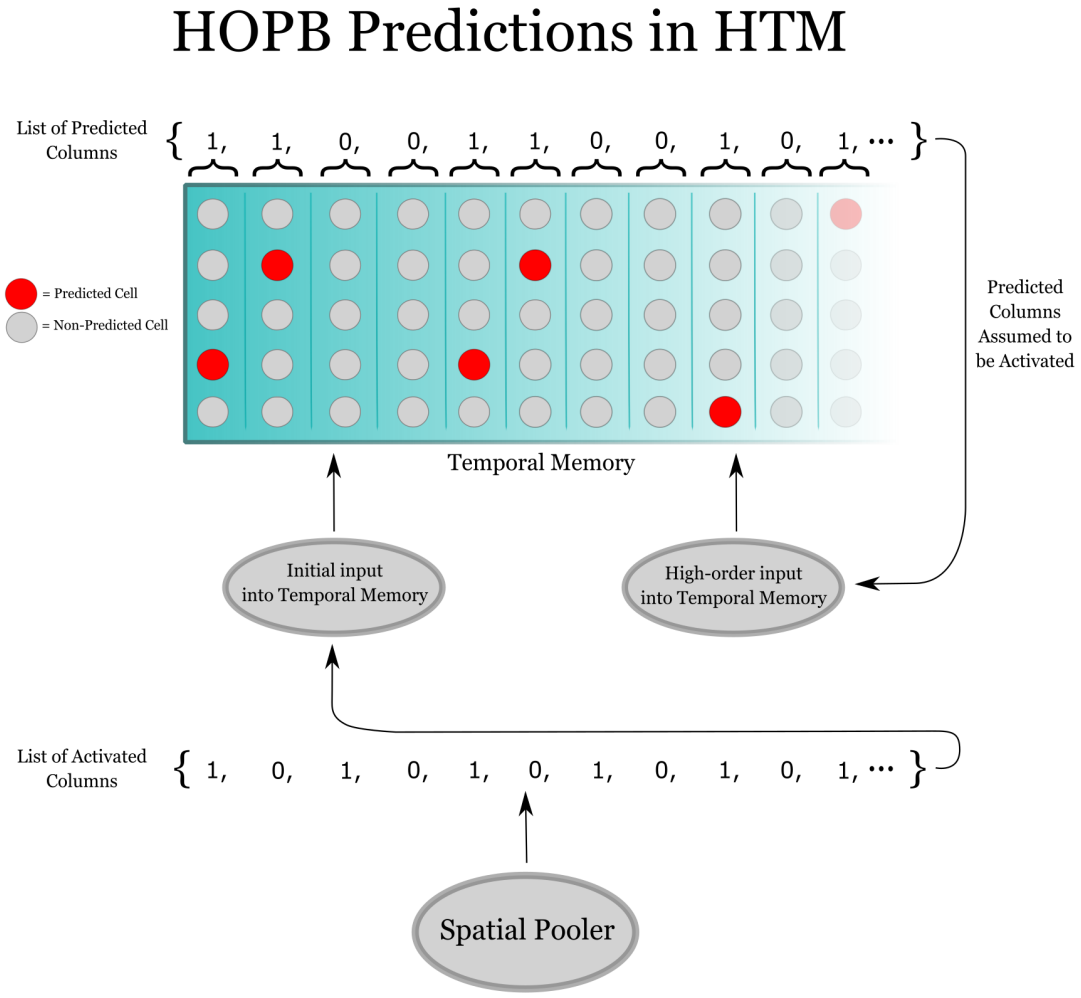

Figure 4.2: *Illustration of the process to get HOPB predictions. The spatial pooler decides which columns to activate in the network. This is the initial input into the temporal memory algorithm. The execution of the temporal memory algorithm with the initial input is executed fully including synaptic permanence updates. The predicted state is then extracted into new column activations and reused as the next timestep input while skipping any synaptic permanence update steps. The cycle of predicted columns to activated columns to predicted columns and so forth can be repeated an arbitrary amount of times.*

In the rest of this thesis, HOPB prediction SDR refers to the set of *columns* that are predicted to become active in the next timestep. This is to be differentiated with the SDR of active or predicted *cells*. The SDR of active cells includes fine-grained knowledge of the status of each individual cell in every column.

Assuming the number of HOPB predictions to make has already been decided, the

procedure for generating HOPB predictions is expressed as pseudocode in the context of the TM algorithm in Algorithm 1. This algorithm computes all the HOPB prediction SDRs for every timestep. The input to the main function, HOPB_Run, is $\mathcal{A}_t$ which is the activated columns initialized as the new incoming output from spatial pooler and n_t which is the number of HOPB predictions to make. Note that at the beginning of the algorithm the current state of predicted cells is the first-order predicted state for the current timestep. This preexisting prediction was generated and already processed in the previous iteration. If it is the first call to this function, the predicted state would simply be empty.

Algorithm 1 Generating HOPB predictions.

1: **function** HOPB_Run($\mathcal{A}_t, n_t$)
2: Process $\mathcal{A}_t$ with the full TM algorithm
3: Π^1_{t+1} = The currently predicted columns
4: I_0 = The current state of the HTM network
5: **for** $i = 2$ **to** n_t **do**
6: Set the active columns to the currently predicted columns
7: Compute the next columns to be predicted
8: Π^i_{t+i} = The currently predicted columns
9: Set the state of the HTM network back to I_0
10: **return** $\vec{C}_t = \{\Pi^1_{t+1}, \Pi^2_{t+2}, \ldots, \Pi^{n_t}_{t+n_t}\}$

In Algorithm 1, each Π^i_j represents the i^{th} order prediction SDR of predicted columns for timestep j. In line 2 we execute the entire temporal memory algorithm, including learning, with the input from the spatial pooler. Π^1_{t+1} is a first-order prediction for the next timestep made with the incoming output from the spatial pooler. Π^1_{t+1} is always the same as the sole SDR used to compute prediction error in Numenta's algorithm described in [12]. Π^2_{t+2} and beyond are strictly high-order predictions. In line 4, we record into I_0 the entire dynamic state of all cells and columns of the network after using the input from the spatial pooler. I_0 represents the state of the HTM network that we want to return to after we're done making HOPB predictions. Lines 5-8 are for producing HOPB predictions. If only first-order predictions are desired, these lines would be skipped. In line 6, we set the new column activations to be the columns currently in a predicted state. Note that to complete this step we need only switch the state of any cell in a predicted state into an active state. Line 7 computes new predicted columns based on the current activated cells which was just updated in the last step. These new predicted columns represent a

high-order prediction and are stored into Π^i_{t+i} in line 8. Line 9 resets the HTM network state in back to what it was immediately after we processed the input from the spatial pooler to continue the real-time stream where learning is constant. Finally, we return each HOPB prediction SDR on line 10.

As an implementation detail, note that while grabbing high-order column predictions (line 7), not only do we not run any synaptic weight adaptations, but we also do not have to run the first phase of the TM algorithm which activates cells based on active columns. We know that the active cells are going to be exactly the predicted cells since we assumed our predictions to be correct. We can simply set the active cells to be the predicted cells. This saves some computation time.

4.3 Dynamic Chain Size

This section presents the design considerations and algorithm for determining a chain size dynamically as the signal progresses.

4.3.1 Limiting Factors

There are several considerations when making HOPB predictions that limit how far into the future is appropriate to predict. These considerations ought to limit the number of HOPB predictions we make at each timestep based on the characteristics of the data and the HTM model itself. These considerations are enumerated below.

1. **Density**: It is well known that density (number of on bits) plays an integral role in the usefulness of SDRs [11]. Accordingly, it should play a chief role in limiting the adoption of an HOPB prediction as acceptable and reliable. Practically, it is recommended to threshold the acceptable density of a HOPB prediction SDR to be between 2% and 10%. 2% is chosen because it is the minimum target density of the spatial pooler by design. 10% is chosen empirically from observing the behavior of first-order prediction SDR density across a wide range of data. In addition, a combinatorial argument is given in 4.3.3.1 that justifies using 10% as a maximum column-wise density. This is to say if an HOPB prediction SDR is generated that has a density outside the acceptable range then we should halt early and stop generated anymore predictions. If the density is too low, the prediction error is likely to be high regardless of the accuracy of the predicted columns. If the density is too high, the prediction error is likely to be low regardless of the accuracy of the prediction. We can interpret an HOPB prediction SDR being too dense as too many possible

futures being predicted at once. Empirical evidence in Section 6.3.1 shows us that HOPB density going to zero (and reporting maximum prediction error) is a bigger threat than overly large SDR densities for typical scenarios. This is hypothesized to be because of the many thousands of connections that exist for each cell and the representational complexity involved in that. These numerous connections are necessary to recognize a network state in many different contexts. Each different pattern and context pair is going to be recognized by a different dendritic branch in general. Thus, it is more likely, especially while the network is early in the learning process, that the lateral basal connections needed to give a high-order prediction have not been reinforced enough yet than too many connections having been turned into active synapses.

Note again the difference between predicted columns and predicted cells. The SDR of predicted cells is not used to compute density because we do not want to increase our sense of density just because several cells in the same column are all in a predictive state. Several cells in the same column all in a predictive state simply means that multiple predicted sequence states share that column in their representations. For the purposes of representing a union of possible futures, this would increase the amount of future information represented while not increasing density of column predictions. Thus, this behavior is desirable.

Acceptable Density Return Frequency Note that by first thresholding HOPB prediction SDRs on density, there arises a related problem of tracking the frequency at which a certain order of prediction returns an SDR with acceptable density. For high orders of prediction, there is no guarantee in general that obtaining a reasonable SDR for timestep t means you will obtain another reasonable SDR for $t + 1$. If a certain order of prediction is unable to return a reasonable SDR with sufficient frequency, there arises a problem where each timestep's final score ends up being an average over a frequently varying number of predictions. When this number of predictions which is used to compute final scores is too inconsistent, the distribution of final scores which is used to compute anomaly likelihoods becomes too unstable and performance suffers. In short, orders of prediction should also be thresholded based on their ability to produce acceptable SDRs with sufficient frequency. We found that requiring an order of prediction to return SDRs with acceptable density at least 80% of the time works well in practice.

2. **Past Prediction Error**: Even if HOPB prediction SDRs have perfectly reasonable

densities with a reasonable frequency, there are still some cases when data is simply too uncertain to be accurately predicted out to a certain point in time. This may be due to noise, natural variability in the signal or a limitation of the HTM model itself. For this reason, we need to incorporate some sense of short-term past prediction error performance of HOPB predictions into deciding a chain size at each timestep.

The method employed and recommended in this thesis is to compare the sample mean plus one sample standard deviation of short-term past prediction errors for a given order of prediction to first-order prediction errors to determine reliability. An average value is an appropriate measure. Even for randomly distributed data, prediction errors tend to have an average value close to zero which is a facet of the temporal memory algorithm itself. The sample mean is also particularly sensitive to outliers (random bursts of error), which is desirable for the purposes of anomaly detection since consistency is key. Adding one sample standard deviation to the sample mean incorporates the spread of errors for an order which is often important for determining the reliability of predictions. We only want to include orders of prediction that are performing approximately the same or better than first-order predictions. An algorithm for autonomously searching for the optimum number of predictions per timestep in a real-time stream is presented in Algorithm 2.

In general, it will not occur that a higher order of prediction consistently gives less error than a lower order prediction except by random chance, so we need only concern ourselves with orders of prediction in an *ascending* fashion. If one order of prediction fails, it invalidates any higher orders of prediction with it. In this thesis, the chain size is made to float up and down automatically. If a given HOPB chain size is reliable for a significant amount of the time, a new higher order of prediction will be tested and included in the chain size if it proves consistent with lower order predictions. The process will continue forever; constantly looking to extend the chain size whenever the HTM model is capable.

Shown in Section 6.3.1.4, we can only truly get a sense of the usefulness of an order of prediction based on the SDRs it produces that are within an acceptable density range. SDRs that do not fall into an acceptable range are not evaluated and do not count against an order of prediction's sample statistics except in that it takes up space in the allowable number of historical records. Section 4.3.2 provides more insight into how we balance the needs of gathering accurate sample prediction errors and accounting for SDRs that do not have an acceptable density.

3. **Inherent Improbability:** Lastly, even if all the HOPB predictions are of stable

density and past prediction error is low, there still exists an inherent improbability with predicting long into the future for any dataset and any algorithm. For example, even if your data is currently following a flat line and is perfectly predictable for many time steps, that doesn't mean we should predict hundreds of timesteps into the future for reasons of inherent uncertainty and computational cost. There is an upper bound on how far out is appropriate. Enforcing an absolute maximum number of HOPB predictions at ten makes sense for a general-purpose application. This keeps the computational complexity of the algorithm tractable and provides sufficient context to better illuminate nearly all types of anomalies.

4.3.2 Getting a Candidate Chain Size

Consistency is key for anomaly detection. Behavior of the algorithm when not experiencing an anomaly will ideally stay consistent else recognizing behavior under anomalous conditions become difficult or impossible. If HOPB is frequently varying the number of predictions for each timestep, we are left with inconsistent behavior. Recall the prediction error score is an average of overlap scores across multiple timesteps. Different numbers of predictions produce different distributions of prediction error scores. Thus, when using HOPB with a dynamic chain size, we wish to discover the optimum number of predictions as quickly as possible and stick with it if possible until change is necessary.

We define the optimum number of predictions for the current data characteristics and HTM model as the largest number of predictions such that:

1. $n_{\max}$ is not breached.
2. Each order of prediction that is called for returns a valid result (acceptable density) with sufficient frequency.
3. Those predictions of acceptable frequency have prediction error statistics that are consistent with first-order predictions.

With these goals in mind, the algorithm for deciding a candidate chain size is presented in Algorithm 2. The main function to call is Calc_N(). We denote the calculated candidate chain size for a timestamp as n_t. We denote the short-term history of prediction errors as H where H is a 2D matrix with a vector of past prediction error records for each order of prediction in the first dimension. H is what is processed and returned in a call of Algorithm 2; n_t is simply the length of first dimension of H at the end of the function. We define the static absolute maximum of HOPB predictions to make as $n_{\max}$.

We denote the number of records to store for an order before measuring return frequency and sample statistics as ψ. We denote the minimum percentage of acceptable density SDRs (return frequency) required in a sample as α. We denote the tolerance to account for sampling error when comparing sample statistics as ϵ. Also as input, we denote the previous timestep's prediction errors as a vector $\vec{e}_{t-1}$ where each element of the vector is the prediction error for an order of prediction. General purpose values for the parameters are $n_{\max} = 10$, $\psi = 500$, $\epsilon = 0.01$ and $\alpha = 0.8$.

Algorithm 2 Calculating chain size based on past performance.

function IS_EXP($\vec{h}$)
    high_order = If $\vec{h}$ doesn't contain first-order predictions
    unreliable = If $\vec{h}$'s size hasn't reached ψ
    if not unreliable **then**
        unreliable = If the percentage of samples is below α
    failed = If $\vec{h}$'s statistic $\gamma_h > \gamma_1 + \epsilon$
    return high_order **and** (unreliable **or** failed)
function CALC_N(H, $\vec{e}_{t-1}$)
    for every $\vec{h}$ in H **do**
        Append associated value in $\vec{e}_{t-1}$ to $\vec{h}$
        if The size of $\vec{h}$ is above ψ **then**
            Remove the oldest value in $\vec{h}$
        if Is_Exp($\vec{h}$) **then**
            Make $\vec{h}$ the last vector in H
            break the loop
    if not Is_Exp(last vector in H) **and** length(H) $< n_{\max}$ **then**
        Add a new empty vector to H
    n_t = length(H)
    return n_t

Note that Algorithm 2 calls for n_t (the number of vectors inside H) HOPB predictions to make but only $n_t - 1$ of the actual HOPB predictions are meant to be used for anomaly detection if not all orders are out of the experimental phase. The last order of prediction is always experimental; trying to increase the chain size whenever it is deemed appropriate unless H is completely full in which case all order predictions are safe to use. A simple Boolean variable on each HOPB prediction SDR is used in the implementation to keep track of the experimental orders of prediction so that their prediction error scores are not

included in the overall prediction error calculation for the timestep. The prediction error scores from experimental orders of prediction are still used, however, to obtain sample data for that order.

The first function, Is_Exp(), is for testing an order of prediction's experimental status. Line 2 ensures the vector isn't meant for first-order predictions, since these must always be used. Line 3 checks if the vector has not reached the desired sample size. Lines 4 and 5 check for, if the vector has reached the desired sample size, if the number of acceptable samples is below the minimum threshold. In the implementation, if an order of prediction returns an SDR that doesn't have an acceptable density, we place a null value into the vector so that it takes up space but doesn't contribute to the sample statistics. Line 6 checks if the vector's sample mean plus one sample standard deviation is above the same statistic for first-order predictions plus a tolerance to account for sampling error. In summary, for an order of prediction to be considered non-experimental, all four of the following must be true.

1. The order of predictions must be greater than one.
2. The size of the sample must have reached ψ.
3. The percentage of acceptable density samples must be above α.
4. The performance statistic must be at or below the performance statistic of first-order predictions plus a small tolerance, ϵ.

Function Calc_N is the main function to call to determine the next timestep chain size. Lines 9-12 update H with the new information in the incoming error vector. This first consists of appending new values to an order of prediction's history in line 10. We always add the newest available information. Line 11 checks if the size of the vector has breached ψ. If it has, in Line 12 we remove the oldest value in the vector to reflect the newest available information.

Lines 13-15 check for any orders of prediction that should be in an experimental phase given the most recent records. Note that an order of prediction that was previously declared stable can fall out of favor in the future do to a variety of factors such as increased unpredictability of the signal. We search for these vectors in an ascending fashion, and if any experimental vector is found, we prune data from any higher orders of prediction. The experimental order of prediction is labeled as such and its historical record continues where it left off. Note that only the highest order of prediction can possibly be in an experimental phase.

Lines 16-17 decide if a new order of prediction should be included (starting in an experimental phase). This only occurs if the highest-order of prediction currently included is not experimental and a new vector wouldn't breach the maximum number of predictions threshold, $n_{\max}$.

Finally, lines 18-19 return the new number of HOPB predictions to call for, n_t, which is the length of the first dimension of H at that time.

Algorithm properties: What we see in Algorithm 2 is an HOPB chain size that floats up and down with the changing characteristics of the data, HTM model prediction quality and past prediction error statistics. HOPB predictions will be utilized to their maximum capacity limited by how trustworthy they have proven to be over time. In other words, the HOPB manager is constantly trying to push more and more orders of prediction to be used but is governed by a self-balancing system which uses the quality of predictions to allow or negate those actions. This is completely dynamic and autonomous which compliments HTM's intrinsic continuous learning properties.

4.3.2.1 Effect of ψ, α and ϵ

The effects that the three HOPB parameters have on performance are relatively straightforward and should be easy to tune across applications. It is recommended that ϵ shouldn't be set past 0.15 as an absolute maximum. If ϵ is any higher, you risk trusting orders of prediction that return bad performance and overall performance will greatly suffer. Using $\epsilon = 0.01$ or $\epsilon = 0.02$ is a reasonable choice for any application if ψ is a reasonably large number.

The primary purpose of ψ is to tell the algorithm how many samples it needs to collect before it establishes an estimate of the sample statistics for that order. See Section 6.4.1 to see that if ψ is too small, the algorithm will often fail to converge to the optimum number of predictions for that dataset's current characteristics and performance will suffer. In general, increasing ψ will provide for more stable behavior when searching for this optimum number. Too large of ψ, however, can make the algorithm slow to adjust to changing characteristics. We can theoretically motivate the minimum value of ψ by considering confidence estimates. Recall the margin of error for the estimate of the sample mean from a sample can be calculated as in Equation 4.6.

$$MOE = t^* \frac{s}{\sqrt{n}} \tag{4.6}$$

Where t^* is the t-statistic, s is the sample standard sample deviation and n is the

number of samples. We know the maximum possible standard deviation of prediction error is going to be 0.5 since every value is bounded within 0 and 1. It would only occur when the samples are perfectly split between the bounds taking on values either 0 and 1. If we assert 95% confidence and assume maximum standard deviation, we can plot Equation 4.6 as a function of the sample size as in Figure 4.3.

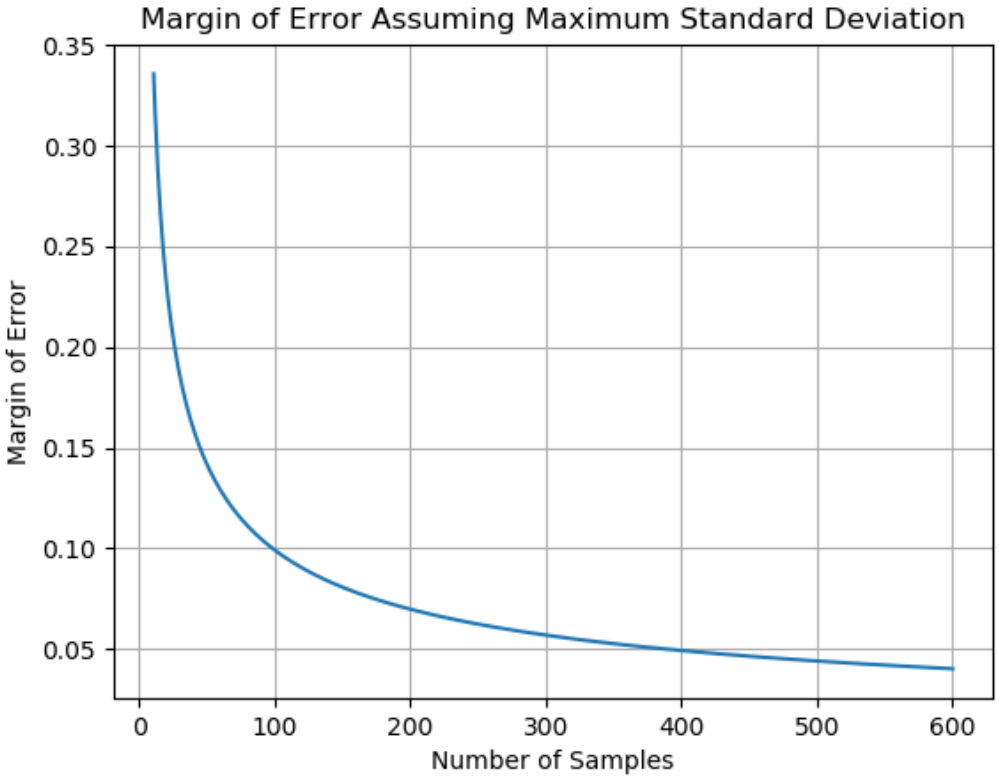

Figure 4.3: *The margin of error of the sample mean estimate asserting 95% confidence and maximum possible sample standard deviation.*

We see in Figure 4.3 that if we use $\psi = 400$ we can ensure our margin of error will not breach 0.05 with 95% confidence. In practice, using a value of ψ that is between 500 and 2000 tends to work well across many applications in terms of finding a stable number of predictions and overall anomaly detection performance. Increments of 500 are good to test with since the algorithm is not sensitive to small changes of this parameter.

Lastly, choosing α is not a difficult process as it should generally just not go below 0.5 or performance will suffer. Good starting points to test with are 0.8 and 0.9 since the algorithm is not sensitive to small changes of this parameter.

4.3.2.2 Note on Efficient Implementation

Calculating the sample mean and sample standard deviation from scratch for each order of prediction every time a record is inserted or deleted in Algorithm 2 is computationally expensive and unnecessary. It is recommended to instead compute the sample mean and sample standard deviation iteratively. This keeps the amount of work to get these measures constant, $O(1)$, per iteration with respect to the number of records stored.

We can accomplish this by keeping track of the sum of all records and the sum of all records squared. Updating these quantities is straightforward when we add or remove a record from the history. Calculating the sample mean iteratively with this quantity is immediately obvious from the definition in Equation 4.7.

$$\mu_n = \frac{1}{n}\left[\sum_{i=1}^{n} x_i\right] \tag{4.7}$$

The computation of the sample standard deviation in an iterative manner is derived in Equation 4.8. One need only apply simple algebraic manipulations to the definition to express it with respect to the iteratively updated variables. For brevity, the derivation is expressed for the sample variance.

$$\begin{aligned}
s_n^2 &= \frac{1}{n-1}\left[\sum_{i=1}^{n}(x_i-\mu_n)^2\right] \\
&= \frac{1}{n-1}\left[\sum_{i=1}^{n}(x_i^2-2\mu_n x_i+\mu_n^2)\right] \\
&= \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2\right] + \frac{-2\mu_n}{n-1}\left[\sum_{i=1}^{n} x_i\right] + \frac{\mu_n^2}{n-1}\left[\sum_{i=1}^{n} 1\right] \\
&= \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2\right] + \mu_n\left(\frac{-2\sum_{i=1}^{n} x_i + \mu_n n}{n-1}\right) \\
&= \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2\right] + \mu_n\left(\frac{-\sum_{i=1}^{n} x_i}{n-1}\right) \\
&= \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2\right] - \left(\frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n(n-1)}\right) \\
&= \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right]
\end{aligned} \tag{4.8}$$

Lastly, note that in practice $\sum_{i=1}^{n} x_i^2$ and $\frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2$ can be very similar numbers. Thus, catastrophic cancellation can lead to the precision of the result to significantly decrease compared to the inherent precision of the floating-point arithmetic used to perform the computation. To avoid this catastrophic cancellation in this formula, we can exploit a location parameter invariance property of the variance. Concretely, for any constant K, Equation 4.9 is upheld.

$$\mathrm{Var}(X - K) = \mathrm{Var}(X) \tag{4.9}$$

This leads to a new formula for the iterative standard deviation shown in Equation 4.10.

$$s_n^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n}(x_i - K)^2 - \frac{1}{n}\left(\sum_{i=1}^{n}(x_i - K)\right)^2\right] \tag{4.10}$$

Using a value of K closest to the population mean is desirable yet any value of K that is within the range of values will suffice. This ensures that the second term in the equation is always smaller than the first and thus no catastrophic cancellation will occur [19]. Note also that, in this case, the prediction errors are always bounded between 0 and 1 thus overflow is not a potential issue.

4.3.3 HOPB with Dynamic Chain Size

The algorithm for generating HOPB predictions with a dynamic chain size is presented in Algorithm 3. This algorithm is to be called after Algorithm 2 has determined the appropriate number of HOPB predictions to make for the current timestep t. Notice the input now is the output of the spatial pooler at timestep t denoted $\mathcal{A}_t$, the minimum acceptable density $d_{\min}$ and the maximum acceptable density $d_{\max}$. The predictions that are made with the incoming spatial pooler activations are always used regardless of their density to establish a absolute minimum number of HOPB predictions returned equal to one. Always using the absolute minimum number of HOPB predictions is equivalent to the Numenta algorithm. Recall general purpose values for the constant density parameters are $d_{\min} = 2\%$, $d_{\max} = 10\%$.

Algorithm 3 Generating HOPB predictions with a dynamic chain size.

1: **function** HOPBwDW_Run($\mathcal{A}_t$, $d_{\min}$, $d_{\max}$)
2: Process $\mathcal{A}_t$ with the full TM algorithm
3: Π^1_{t+1} = The currently predicted columns
4: I_0 = The current state of the HTM network
5: n_t = Calc_N()
6: **for** $i = 2$ **to** n_t **do**
7: Set the active columns to the currently predicted columns
8: Compute the next columns to be predicted
9: **if** $d_{\min} \leq$ Current predicted column density $\leq d_{\max}$ **then**
10: Π^i_{t+i} = The currently predicted columns
11: **else**
12: Π^i_{t+i} = A null value
13: Set the state of the HTM network back to I_0
14: **return** $\vec{C_t} = \{\Pi^1_{t+1}, \Pi^2_{t+2}, \ldots, \Pi^{n_t}_{t+n_t}\}$

The main difference from Algorithm 1 is in line 9-12 which actively monitor HOPB prediction SDR density. The chain size is also governed by n_t in line 6 which is determined in line 5 where we execute the process in Algorithm 2. All the relevant data about past prediction error performance per order is assumed to be available. Note that this algorithm is dynamic and adjusts the chain size with the data in real time. If the dataset becomes more predictable later in the stream and more HOPB predictions can be made (which provides more sequence context), the algorithm will dynamically adjust to that and the converse.

4.3.3.1 Understanding the Parameter $d_{\max}$

The decision on acceptable maximum column-wise SDR density can be seen as choosing the acceptable probability of a false match on a dendritic segment which may vary from application to application. We say a dendritic segment recognizes a pattern, i.e. undergoes an NMDA spike, if at least θ of its s synapses are connected to the set of cells that are active at the same point in time. Consider a population of n cells with a of them active at any given time. Assuming a random distribution of patterns and $a << n$, the probability of a false match can be calculated as follows [46].

$$\frac{\sum_{i=\theta}^{s} \binom{s}{i} \times \binom{n-s}{a-i}}{\binom{n}{a}} \tag{4.11}$$

It's this parameter, a, which that has the potential to grow out of control with HOPB predictions. Therefore, thresholding a with $d_{\max}$ is effectively thresholding the probability of a false match on a dendritic segment. This might provide for a much more intuitive parameter.

We know how to exactly calculate a for each HOPB prediction iteration because the activated columns are exactly the predicted columns. This means the activated cells will be exactly the predicted cells. Thus, for each HOPB prediction SDR which contains the predicted and activated columns for that timestep, we can compute the density of Π_c which we will call the SDR of *cells* in a predicted state across the entire population. Computing a is now as easy as Equation 4.12.

$$a = |\Pi_c| \tag{4.12}$$

Where $|\Pi_c|$ is the scalar norm, i.e. the number of on bits, within Π_c. This provides a more intuitive way to systematically and dynamically decide what $d_{\max}$ should be for your application. If, for a given HOPB prediction SDR, we find that the risk of a false match as computed in Equation 4.11 and Equation 4.12 is above a certain threshold, we discard it and return the current collection of HOPB prediction SDRs.

Note that this probability will be tiny for typical parameter settings even when assuming the maximum column-wise density of 10% occurs. Consider a network of 2048 columns with 32 cells each which is a total population of $n = 65536$ cells. Assume each dendritic segment has $s = 10$ synapses and the threshold to activate is $\theta = 10$. A 10% sparse column-wise SDR means that 205 columns are in a predicted state (rounding up) and if we assume the worst-case scenario, that means all 32 cells in each of those 205 columns are in a predicted state (although this will essentially never happen). However, if it *did* happen, that means we would have $a = 6560$ cells activated. Equation 4.11 tells us that each dendritic segment in this case has a 1.003×10^{-10} probability of accidentally being activated by a different pattern that shares the cells it's connected too. If we assume each cell in the network has 10 dendritic segments, that brings the total probability of a false match on a singular dendritic segment for that timestep to 6.576×10^{-5}. Note that several dendritic segments would need to be falsely activated to cause significant changes in the resulting predicted column SDR thus that number is the chance of any, even insignificant amount of accidental activations. We know, however, that all 32 cells in each activated column being in a predicted state is incredibly unlikely. Much more reasonable is to assume maximally 2 cells per activated column are in a predicted state. Thus, a much more reasonable estimate of a would be $a = 410$. Using this brings Equation 4.11 down to

8.227×10^{-23} and the total probability of any false dendritic activation anywhere in the network to only 5.392×10^{-17}. Thus, using $d_{\max} = 10\%$ is a reasonable choice and does not significantly increase our chance of false dendritic activation when calculating HOPB prediction SDRs.

Note that in the case of disappearing SDR density, the probability of a false match goes to zero, but the prediction is useless. Thus, $d_{\min}$ should not be determined this way. It is recommended to always keep $d_{\min}$ at 2% since that is approximately the minimal desired density coming from the spatial pooler under normal parameter setups.

4.4 Using HOPB Predictions

Note that we create a set of HOPB predictions for every timestep in the real-time stream as soon as it becomes available. We limit all learning mechanisms in HTM to only occur during single order transitions at each new timestep. In Algorithm 3, we see that, before we exit, the state of the TM region is returned to what it was before any HOPB predictions have been made. Thus, a real time stream where learning is constantly active is being executed on the temporal data in the regular sequence. At each timestep, we are first processing it in the same way as Numenta's algorithm but then additionally extracting high-order sequence prediction information in a way that does not alter the real-time stream.

After HOPB predictions have been obtained, we need to use them for streaming anomaly detection. This demands the determination of a timestep as anomalous or non-anomalous precisely when it occurs and *without* any future data. Obviously, HOPB predictions are for future timesteps, so the determination of a timestep will hinge on HOPB predictions made in the past.

4.4.1 The HOPB Manager

There exists a need to manage all HOPB prediction SDRs efficiently. Any HOPB chains beyond size one must be remembered for multiple timesteps as the previously future predictions are being evaluated. HOPB chains will constantly overlap and will frequently fluctuate in size in general. Matching HOPB prediction SDRs to data values by timestamps is costly and involves lots of searching which introduces computational costs which need not be spent. Instead, this thesis introduces a custom data structure to manage all HOPB predictions simultaneously. The data structure is organized into an $n_{\max} \times n_{\max}$ 2D matrix of SDRs where $n_{\max}$ is the absolute maximum number of HOPB predictions to

make in a timestep combined with a single pointer and a variable for the current spatial pooler activations. This data structure is illustrated in Figure 4.4.

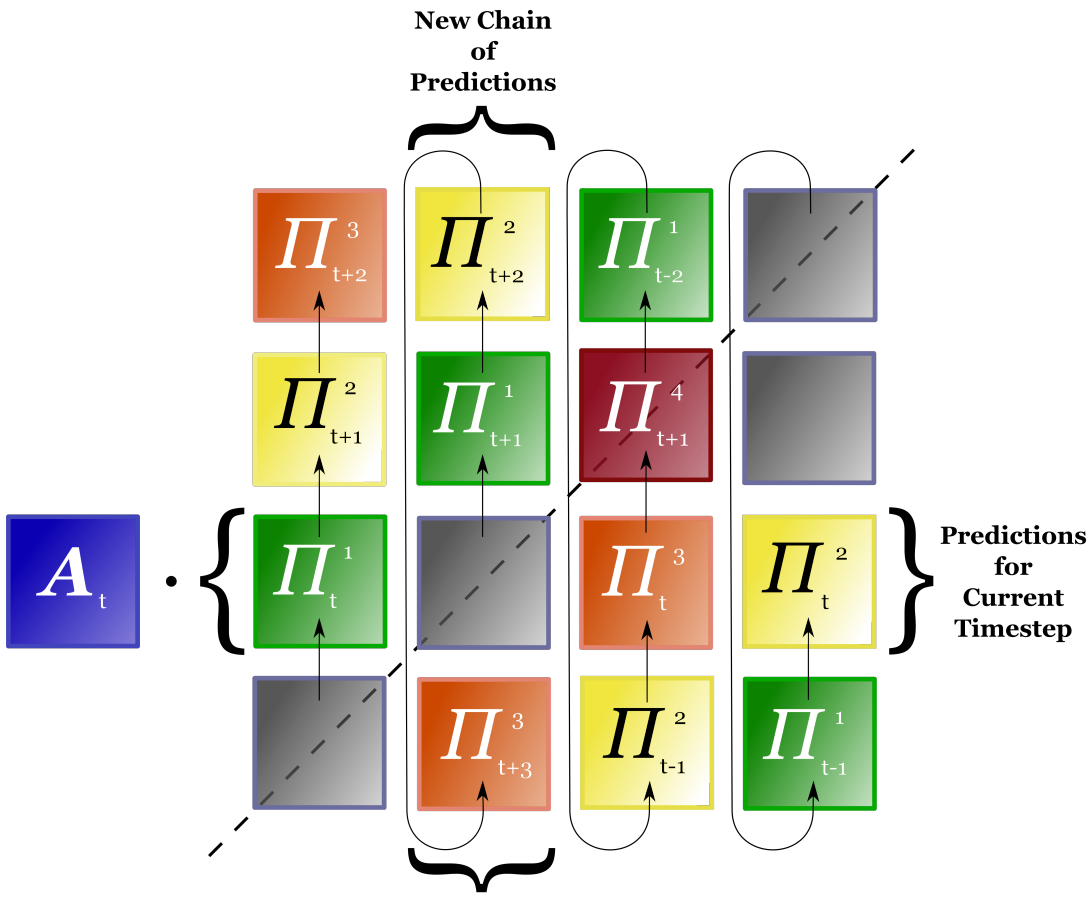

Figure 4.4: *Visualization of the HOPB manager data structure. A single pointer iterates through the diagonal entries with each timestep (while looping around) and defines the current row and column of interest. Slicing horizontally gives us the HOPB prediction SDRs for the current timestep each made at a different point in time. The current spatial pooler activations are evaluated against the SDRs in the current row of interest. Note that old or missing data might exist in some of these row entries. We need only evaluate and collect scores from entries which do not have a score yet to solve this problem. Slicing vertically gives us the HOPB prediction SDRs for a chain which originates at the diagonal entry. The new chain of HOPB prediction SDRs is added to the column, overwriting previous information, after scores have been collected for that timestep. The number of HOPB prediction SDRs per chain can fluctuate freely through time bounded only by an absolute minimum of 1 and an absolute maximum of* $n_{\max}$.

By maintaining a single pointer that represents the current row and column (diagonal

entry) of interest, we keep track of where in the data structure lies the previous predictions and where a HOPB prediction chain of SDRs should be written. First, for a given timestep, the new SDR coming from the spatial pooler which represents the actual column activations can be compared to all the HOPB prediction SDRs across the current row of interest. That row of HOPB prediction SDRs will have scores computed and each SDR represents a guess made at a different point in time. We extract all the scores in the row that we compute in this step to obtain a vector of scores for a single timestep composed of guesses made at various points in time. Then, the new HOPB prediction chain of SDRs is written into memory one row above the pointer extending upward and wrapping around to the bottom. When those two steps are done, we increment the pointer once and use modulo division to wrap it around to zero if necessary. This continuously overwrites and uses memory efficiently to manage all HOPB prediction chains simultaneously. Any kind of search is completely eliminated when using this data structure; We know exactly where each HOPB SDR should be located (if it exists) in memory and can access it in $O(1)$ time. Additionally, no memory is kept around for longer than it's needed.

Note that it is possible for old data to exist in certain spots in the row when computing scores that weren't meant for the current spatial pooler activations. We need only collect those scores that aren't initialized yet to solve that problem. Additionally, we can infer when each prediction was made for each prediction in the row based off the structure of prediction placement which remains constant through execution. A first-order prediction for the current spatial pooler activation SDR will always exist immediately one column to the left of the pointer and they increase one order of prediction for each column you move left while wrapping around with modulo division until you wrap around to the beginning.

Note also that we may determine scores for an individual chain by traveling vertically across a column once all the scores in that chain have been computed. To do this, we need only augment each SDR entry with additional flags which tells us if it marks the beginning or the end of a HOPB chain. Vectors of predictions composing a chain are currently not utilized in the framework in any way, however.

4.4.2 Combining HOPB predictions

We use the HOPB manager to obtain a vector of scores that represent the prediction scores of the current timestep made at various points in the past. Due to the strict requirements imposed on the quality of HOPB predictions, we can be relatively certain that any large value of prediction error, whether from first-order or higher-order predictions, is signaling novelty in the data signal. It turns out that, shown in Section 6.5.4, even if a mistake is made at a certain order, the other orders of prediction for that timestep will typically

dilute that error from effecting the results.

The new calculation of prediction error of a timestep includes multiple prediction error scores each made at a different point in the past. In this thesis, we combine the information of all HOPB predictions for a timestep with a simple averaging procedure. Concretely, we denote an HOPB prediction SDR for timestep t made in the past at timestep $t-i$ as Π_t^i. Given a set of n HOPB prediction SDRs for timestep t, $\{\Pi_t^1, \Pi_t^2, \ldots, \Pi_t^n\}$, and the SDR of true column activations for the timestep from the spatial pooler, $\mathcal{A}_t$, we calculate the final prediction error score for the timestep, $\bar{s}_t$, as in Equation 4.13.

$$\bar{s}_t = \frac{\sum_{i=1}^{n} 1 - \frac{\Pi_t^i \cdot \mathcal{A}_t}{|\mathcal{A}_t|}}{n} \tag{4.13}$$

Recall the $\cdot$ operation is to calculate the number of bits of overlap between the two SDRs and $|\mathcal{A}_t|$ is simply the number of on bits in $\mathcal{A}_t$. Note that due to the chain size dynamically floating up and down, a timestep is not guaranteed to have HOPB prediction SDRs originating from all timesteps between $t-1$ and $t-n$. This is okay and won't negatively impact the results. In general, the more context the better as long as the limiting factors described in Section 4.3.1 are obeyed.

4.4.3 Accumulating Average

Note that HOPB predictions alone cannot capture the presence of accumulating error over time. Each timestep has additional context included inside of its prediction error calculation but each timestep is evaluated independent of the rest under this setup. Additionally, HOPB is designed to distribute the prediction error of an anomaly over all timesteps that it is composed. To account for this, we do one last step before we report the final score to the anomaly likelihood model. We will denote the final reported score for a timestep as f_t. In order to get the desired effects of accumulating error over time, we average the current averaged timestep prediction error score, $\bar{s}_t$, with the final score of the previous timestep f_{t-1} as in Equation 4.2.

This means only under repeated exposures of high prediction error will f_t rise to significant values. This may seem simple, but it has a powerful effect that can be viewed in Section 6.5.4. On the same token, it allows for collective anomalies which cause sequential repeated bursts of error to be better isolated from non-anomalies as seen in Section 6.5.3. In short, using an accumulating average helps better pick out events of high prediction error that have the most evidence of being anomalous.

4.4.4 Computing Anomaly Likelihood

Thresholding on the final scores alone is not advisable since in any dataset there is going to be an inherent level of error that is normal. This is dependent on both the level of natural unpredictability (noise or randomness) present in the data and the error associated with the model as it is continuously learning. For anomaly detection, we want to be able to detect significant changes in the prediction error scores with respect to what is normal for the dataset.

The solution proposed in [77] and the Numenta algorithm [12] is to model the distribution of prediction errors over time. We use a rolling normal distribution which is also used in the Numenta algorithm. This is to incrementally update the sample mean and variance of a normal distribution based on a past window of prediction errors. More details about this method are discussed in Section 3.1.6. This gives us the probability that f_t represents an anomaly, denoted $P(f_t)$.

In practice, we want to better discern between high levels of the anomaly likelihood thus we transform the probabilities through the following function $\mathcal{F}$ in Equation 4.14.

$$\mathcal{F}(P(f_t)) = \frac{\log(1 - P(f_t))}{\log(1 - 0.9999999999)} \tag{4.14}$$

This "log likelihood" simply transforms the measures to a scale that we can more easily see small differences in large probabilities. It is this log likelihood measure that is thresholded to tag or not tag the timestep as anomalous. We use the same procedure in the proposed framework to measure anomaly likelihood, but we are instead feeding in a more robust, context-aware measure of prediction error instead of instantaneous transitional prediction errors.

Chapter 5

Methodology

In this section, we will present the concrete hypothesis that this thesis explores as well as the evaluation methods used to confirm or deny it.

5.1 Hypothesis

It is hypothesized that the method of obtaining HOPB predictions with HTM combined with the surrounding framework for anomaly detection, as described in Chapter 4, will provide both sequence context and a sense of accumulated error into the calculation of prediction error. It is hypothesized that this will help anomaly detection with HTM better identify complex contextual and collective anomalies as well as benign noise while not forfeiting the ability to detect short-term spatial anomalies in a timely manner. Additionally, using HOPB is hypothesized to increase the ability to tolerate faults from the underlying HTM model to provide for more reliable anomaly detection performance.

5.2 Evaluation

The evaluation methods that I will utilize to test the hypothesis in Section 5.1 include a host of custom experiments designed to verify the motivations and intuitions of the proposed framework, the reporting and discussion of comparative algorithm behavior during specific illustrative examples of anomaly models from various real-life external datasets originating from a variety of domains as well as a more general and expansive comparative evaluation benchmark known as the Numenta Anomaly Benchmark (NAB) [64]. In the effort to reduce the impact associated with randomly initializing the HTM network, we will use several different seeded initial randomizations of the underlying HTM model and,

for each, execute the Numenta algorithm with and without and the proposed framework against NAB separately. This will create data from which we can analyze the statistical significance of the results. We wish to show that using HOPB provides a statistically significant benefit in the results regardless of the underlying HTM model.

The proposed framework will be compared to the Numenta algorithm for streaming anomaly detection as proposed in [12] as well as eight other popular streaming anomaly detection algorithms and two control algorithms implemented inside NAB. These eight other algorithms are listed in Section 5.2.2.3. A brief description of each algorithm as well as their citations where applicable for further information can be found there as well.

5.2.1 Verification Experiments

There are several custom designed verification experiments that will be performed to test the efficacy of HOPB in different ways. These types of experiments are enumerated below.

1. **Verification of Motivations**: These experiments are designed to test the motivation behind this thesis. Namely, the inability of the Numenta algorithm to sufficiently capture sequence context and accumulating error in its measure of prediction error per timestep which are necessary to detect especially complex temporal anomaly models. To demonstrate this, a sine wave dataset has been created and three individual anomaly models have been manually inserted. These anomaly models include a short-term spatial anomaly, a contextual anomaly and a collective anomaly as defined in Section 2.1.1.2. The behavior of the Numenta algorithm under these circumstances is observed and discussed.

2. **HOPB Verification Experiments**: In order to test and verify the potential of HOPB to satisfy the hypothesis as well as motivate the design considerations, we include the following experiments.

   Density Experiments: These experiments were set up to monitor and analyze the HOPB prediction SDR density when obtained at increasingly higher orders. We observe the HOPB prediction SDR density at various orders of prediction when the algorithm is exposed to randomly generated data as well as predictable data at increasing levels of noise to simulate increasing unpredictability. These experiments explore the potential benefit of thresholding HOPB prediction SDR density to determine acceptable predictions by showing the relative distribution of prediction error with and without thresholding on density.

   Chain of Context Experiments: This experiment was set up to investigate

if HOPB predictions have the potential to capture high-order sequence information and thus contain useful information to help satisfy the hypothesis. To investigate this, we introduce a context shift in a sine wave and record the high-order predictions made one timestep before the shift occurs. We then measure the prediction error with respect to the spatial pooler activations that occur from the original sine wave and the contextually anomalous data. This experiment is repeated under various levels of noise as well.

3. **Dynamic Chain Size Experiments**: These experiments are intended to explore the effect of using a dynamic chain size on the overall behavior of the algorithm.

   Searching for Convergence: This experiment visualizes the behavior of Algorithm 2 as it searches for the optimum number of predictions to make per timestep under various conditions. A relatively long sine wave is fed in under various levels and models of noise to test convergence behavior. Parameter settings are also explored to visualize their effect on convergence.

   Visualization of the Benefit: This experiment tracks the prediction error performance over a full dataset to illustrate the benefit of using a floating chain size versus a static declaration of chain size.

4. **HOPB Behavior Experiments**: Included here are several experiments to visualize the behavior of HOPB and what it provides for anomaly detection. In here we repeat the same experiments used to verify the motivations of the thesis but use HOPB instead of the Numenta algorithm to see the different behavior. Additionally, we test and witness the random fault canceling effects of using HOPB and what it means for anomaly detection.

5. **Comparative Performance on External Datasets**: Several more datasets from various external sources will be isolated and evaluated for comparative performance to increase the credibility of the results. See Section 5.2.3 to find details about these external datasets.

5.2.2 Numenta Anomaly Benchmark

The Numenta Anomaly Benchmark (NAB) was first conceived in 2015 to provide an unbiased performance benchmark for streaming anomaly detection algorithms in [64]. NAB's intention is to provide a method of comparing streaming anomaly detection algorithms in a way that is more informative than simple false positive and false negative ratios. For

instance, scoring mechanisms not meant for streaming algorithms don't incorporate the value of detecting an anomaly early.

In full, NAB contains many datasets from various sources and implements a custom scoring mechanism described more below. Each dataset is a time-series record of values with a single numeric value per timestep. Each dataset in NAB is assumed to be a single signal from a single source for the sake of simplicity. Note that multi-signal and multi-sensor data can be easily accommodated with the proposed algorithm. For discussion on how to do this, see Section 7.2.

5.2.2.1 Scoring Mechanism

NAB contains a custom scoring mechanism for streaming anomaly algorithms to determine overall performance under three possible application profiles. To score real-time anomaly detectors, we define the requirements of an ideal streaming anomaly detection algorithm as the following:

1. It detects all anomalies present in the streaming data.
2. It detects those anomalies as soon as realistically possible.
3. It triggers no false alarms.
4. There is no manual parameter tuning needed during execution and between datasets. Any data-specific parameter tuning must occur online without any human intervention.

Points 1-3 are obvious qualities of an ideal streaming anomaly detection algorithm. Point 4 is included because of the unique challenges that streaming situations present. Streaming data is dynamic and often experiences drifting data characteristics for any number of reasons. Ideally, the algorithm should be able to adjust to these changes automatically to accommodate any number of unexpected scenarios while minimizing the need for any human intervention. The scoring mechanism is designed to reward algorithm performance according to these criteria.

Concretely, the scoring mechanism consists of anomaly windows and a scoring function. To promote early detection, anomaly windows are a range of timesteps centered around a ground truth anomaly label. Each anomaly tag that an algorithm makes can either positively impact the total score (if true positive) or negatively impact the total score (if false negative or false positive). True negatives do not count anything toward the score. The raw scores take values between -1 and 1. The positive score obtained for

correctly tagging the anomaly is determined by where in the window the first tag occurs. If multiple tags occur in a window, only the earliest tag is used. The score received is a scaled sigmoid value that rewards predictions that occur earlier. False positives are also punished in correspondence with where they occur. If they occur close to an actual anomaly but still outside the window, after the anomaly has occurred, the magnitude of the negative score will be less. The scaled sigmoid value for each tag that isn't ignored as a later tag in a window is then multiplied by a weight determined by one of three application profiles and then summed together to reach a final score. Formally, the computation for a scaled sigmoidal score is shown in Equation 5.1.

$$\sigma^A(y) = (A_{TP} - A_{FP}) \left(\frac{1}{1 + e^{5y}} \right) - 1 \tag{5.1}$$

Where A is the application profile, A_{TP} is the weight of true positives, A_{FP} is the weight of false positives and y is the relative position in the anomaly window. The parameters are set such that $\sigma(0) = 0$, the maximum score is A_{TP} and the minimum score is A_{FP}. Note that $0 \leq A_{TP}, A_{TN} \leq 1$ and $-1 \leq A_{FP}, A_{FN} \leq 0$. Missing a window entirely constitutes a false negative and incurs a score of A_{FN}.

The score for a data file d is then a summation of the scores obtained for each window in d in Equation 5.2.

$$S_d^A = \left(\sum_{y \in Y_d} \sigma^A(y) \right) + A_{FN} f_d \tag{5.2}$$

Where f_d is the number of false negatives obtained. The score over all data files, S^A, is simple the summation of scores obtained for each data file. Then, finally, we obtained the final normalized score, $\dot{S^A}$, by using the maximum obtainable score, $S^A_{perfect}$, and the score of a null detector which doesn't make any predictions, S^A_{null}. This is illustrated in Equation 5.3.

$$\dot{S^A} = 100 \left(\frac{S^A - S^A_{null}}{S^A_{perfect} - S^A_{null}} \right) \tag{5.3}$$

The three scoring profiles reported by default in NAB reflect the different interests of different applications with respect to false positive and false negative generation. For instance, in computer security, low false positive generation is often the highest priority. Alternatively, false negatives might be very costly to industry when monitoring the health of machinery. The standard profile assigns relative weights such that a random detector making random anomaly tags 10% of the time would get a final score of zero on average.

The reward low false positive profile assigns greater penalty to false positives and the reward low false negative profile assigns greater penalty to false negatives.

5.2.2.2 NAB Datasets

NAB contains a variety of datasets that are used to compute NAB scores. These datasets include both synthetic and real-life data aimed to capture a variety of anomaly models. As of version 1.0, NAB contains 58 datasets which range in size from 1,000 to 22,000 timesteps. These datasets are organized as follows.

1. **AWS Server Metrics**: Several datasets collected by the Amazon Cloudwatch service including signals such as CPU Utilization, Network Bytes In, and Disk Read Bytes.
2. **Ad Exchange**: Online advertisement clicking rates over time.
3. **Known Causes**: A variety of datasets where the cause of the anomaly is known. Examples include ambient temperature in an office system during a system failure, timing of key strokes for several users of a computer where anomalies represent a change in the user and CPU usage data from a server in an Amazon datacenter during a system failure.
4. **Traffic Data**: Real-time traffic data including metrics such as occupancy, speed and travel time.
5. **Tweets**: A collection of twitter mentions over time of large, publicly-traded companies such as Google and IBM.
6. **Artificial Data**: Several artificial datasets intended to illustrate different kinds of anomaly models under various noise conditions.

5.2.2.3 Algorithms For Comparison

The following algorithms are included for comparison against the NAB benchmark.

1. **Numenta**: See Section 3.1.6 for a description of the Numenta algorithm and relevant citations.
2. **CAD OSE**: Contextual Anomaly Detector Open Source Edition. This algorithm explicitly stores small blocks of context in a dictionary and matches them to new timesteps to predict the future. Unfortunately, no citations are available for this

algorithm. The code can be found in the NAB repository. This was originally an entry in a NAB organized competition in 2016 [90].

3. **KNN-CAD**: An application of the multi-dimensional conformal anomaly detection on time-series algorithm [16]. A concise description of this algorithm can be found in [53]. This was originally an entry in a NAB organized competition in 2016 [90].

4. **Relative Entropy**: An implementation of online anomaly detection using a relative entropy statistic with multiple hypotheses as described in [125].

5. **Twitter ADVec v1.0.0**: Uses something called a Seasonal Hybrid ESD (S-H-ESD) which builds upon the Generalized ESD test [110] for detecting anomalies in the presence of long-term trends and seasonality [121].

6. **Etsy Skyline**: See Section 3.2.3 for a description of Skyline and relevant citations.

7. **Windowed Gaussian**: A sliding window detector that computes anomaly score of a data point by computing its probability from a Gaussian distribution over a window of previous data points [64].

8. **Bayesian Online Changepoint Detection**: An implementation of the online Bayesian changepoint detection algorithm as described in [10].

9. **EXPoSE**: See Section 3.2.3 for a description of EXPoSE and relevant citations.

10. **Random Detector**: This detector makes random guesses and acts as a control algorithm.

11. **Null Detector**: This detector always guesses no anomaly and acts as a control algorithm. This algorithm's scores are used to normalize the scores of every other algorithm to be between 0 (this algorithm's score) and 100 (best possible score).

5.2.2.4 Current Issues with NAB

As of NAB version 1.0, there exist some concerns and challenges with the benchmark which I will enumerate here. The work in [115] outlines some points of concern about NAB. Interestingly, Figure 1 inside [115] is actually based on a misunderstanding of how the scaled sigmoid function is calculated. Several other points brought forth in the paper are based on debatable premises. However, the work does bring fourth some other valid concerns which we list below mixed with personal observations.

1. Determining the anomaly window size is an arbitrary process. There does not exist a systematic way to choose these window sizes in part due to the potentially ambiguous nature of anomaly detection itself in certain cases.

2. In Equation 5.1, we see the term e^{5y} is a part of the scaled sigmoid but it is not made clear why the number 5 is chosen. It is unclear what coefficient is the "best" one to provide a realistic and unbiased scoring process. The extent of the effect using other numbers would have on the final comparative results is not explored.

3. Some of the datasets contained in the NAB framework contain missing values. These problematic datasets contain values collected at varying intervals and no preprocessing techniques are used to make sense of the missing values other than concatenating the data into one series with varying length intervals. These varying length intervals would naturally hinder any time-series modeling of the underlying signal.

4. Several of the datasets contained in the NAB framework are exceedingly small especially for the underlying assumption of judging algorithms built for streaming environments where new data is constantly arriving. Even more, several datasets in NAB also contain transforming data distributions. By itself, having transforming data distributions within at least some of the benchmark datasets is desirable to test any algorithms ability to adapt to those changes, but this combined with the small size of some datasets sometimes provides for insufficient opportunities to recognize normal behavior of the signal even for humans. These properties of the data virtually eliminate the possibility to test against supervised models such as Long-Short Term Memory networks.

5. The optimization algorithm in the NAB framework determines the threshold on final scores from which to tag anomalies for each algorithm. The optimization algorithm is executed for each application profile separately to maximize the score for that application profile. Concerningly, the optimization algorithm is a local hill climbing technique and is not guaranteed to find the globally optimal solution which may accidentally underestimate an algorithm.

For these reasons, it is recommended in this thesis to take the reported NAB scores with a grain of salt until the above concerns are addressed with later versions of NAB. The reader should consider the NAB scores only in combination with the other evaluation methods.

5.2.3 Isolated Datasets

The four artificial datasets presented in Section 6.2 were created specifically to help illustrate an argument. The proposed algorithm will be executed on these datasets to be able to illustrate the main difference in capabilities between it and the Numenta algorithm as clearly as possible. This thesis additionally provides comparative results from external datasets obtained from various third-party sources. These datasets represent a wide variety of potential application domains. Important characteristics about the five datasets are summarized in Table 5.1.

Table 5.1: *Summarized information about the important characteristics of each third-party dataset comparatively tested in this thesis.*

Dataset	**Domain**	**Anomaly Cause**	**Anomaly Type**	**Number of Timesteps in Subsample**	**Sampling Frequency**
Electromyogram Readings	Healthcare	Myopathy	Context Shift	5170	0.25 milliseconds
Yahoo! Production Traffic	Networking	Usage Spike	Spatial	1420	1 hour
Arterial Pressure Readings	Medical Research	Premature Ventricular Contraction	Spatial/ Contextual	5899	0.2 seconds
Annual Common Stock Price	Finance	Historical Causes	Spatial/ Contextual	99	1 year
Network Health	Cybersecurity	Smurf Attack	Spatial/ Contextual	13515	Non-uniform

More detailed information about each of datasets is discussed below.

1. **Electromyogram**: The first dataset consists of readings from an electromyogram (EMG) which are tests to record the electrical activity of muscles. EMGs can be used to detect abnormal electrical activity of muscle that can occur in many diseases and conditions including muscular dystrophy, muscular inflammation, pinched nerves and peripheral nerve damage [100]. The exact dataset comes from an EMG-focused subset of the publicly available PhysioNet database [8]. The data comes from a 57-year-old male with myopathy due to a long history of polymyositis. Concretely, a 25mm concentric needle electrode was placed in the tibialis anterior muscle of the patient. The patient then dorsiflexed the foot gently against resistance, then relaxed while readings were taken place. The entire dataset consists of many readings taking place over only a few seconds. There is a short cease of electrical activity from the

muscle late into the time-series which is characteristic of myopathy. This is the temporal anomaly we wish to discover. The dataset and the results can be seen in Section 6.6.1.

2. **Yahoo! Production Traffic**: The second external experiment is an isolated dataset from the Yahoo! Webscope S5 dataset [61] made publicly available through the Yahoo! Webscope^{TM}Program. We include an example of one of the datasets consisting of production traffic metrics on some of the Yahoo! properties. The dataset features an example of spatially anomalous traffic behavior. Note that the datasets inside this benchmark include labels, but as HTM is an unsupervised algorithm, we did not use them in any capacity. The dataset and the results can be seen in Section 6.6.2.

3. **Arterial Pressure**: Third is a dataset consisting of arterial pressure readings in rats generously shared by the Department of Physiology at Michigan State University [67]. The rats are anesthetized, and their blood pressure was monitored during various forced stimulation exercise bouts to monitor the driving pressure for blood flow to the exercising muscle. Any changes in heart rate (time between pressure peaks) or elevation in pressure is in response to hindlimb stimulation. The provided dataset is quite large: 255,410 timesteps taken at 0.2 seconds intervals. Concretely, the mean pressure in mmHg (millimeters mercury) is recorded every 0.2 seconds. It is not uncommon to see very large dataset sizes in streaming environments such as this especially when the sampling interval is short. It would be very time consuming for a human to sift through all this experimental data looking for unusual behavior. In this specific subset of the larger dataset, the rat experiences a premature ventricular contraction which causes anomalous behavior in the blood pressure readings. The dataset and the results can be seen in Section 6.6.3.

4. **Annual Common Stock Price**: Fourth is a dataset from the financial domain that is publicly available at the Time Series Data Library [51]. We test HOPB on a record of the annual average common stock prices in the United States from 1871 to 1970. This data has a sampling frequency of one whole year. This dataset accordingly contains only 98 entries and tests the frameworks ability to work with exceedingly small datasets. Concretely, we are interested in two anomalies that occur with the average common stock price over that period. The first is the bubble that occurs during the 1920's when the stock market began to boom followed by the crash of 1929 which led to the Great Depression. The second anomalous behavior is captured in the sharp increase in stock prices that took place from approximately 1950 to 1970.

In this case example, we show that even with a shortage of data, HOPB helps to distinctly and properly capture the two individual anomalies. Note that HOPB may be employed on any financial data with virtually any sampling frequency limited only by processing time which is explored in Section 6.8. The dataset as well as the results can be seen in Section 6.6.4.

5. **Cybersecurity**: We've gathered a subset of the widely studied KDD-Cup '99 Dataset [111]. This dataset contains aggregated statistics and derived features about connections to a simulated military network environment. The original goals of the dataset included constructing a system capable of classifying individual connections to a network as benign or malicious. The full dataset contains a wide variety of simulated intrusions as well as normal traffic. In our experiment, we isolate an event known as a "Smurf" attack which is a member of a larger class of attacks known as distributed denial of service (DDoS) attacks [69]. The Smurf malware sends a spoofed source IP network packet that contains an Internet Control Message Protocol (ICMP) ping with a broadcast IP destination. The resulting echo responses to the ping message are delivered to victims IP address. The goal of a Smurf attack is to flood a target IP address with enough spoofed ICMP ping messages that it becomes overloaded and often crashes. Naturally, the derived traffic feature in each record we use to capture this attack is the number of connections to the same host as the current connection in the past two seconds. The records do not have any timestamps and are not collected at equal intervals. We simply place some records of normal network connections surrounding a group of records labeled as a part of a Smurf attack and read them sequentially. We make no use of the labels or the provided training and testing divisions since the proposed framework is completely unsupervised and need only make one pass through the data to recognize the attack. The dataset as well as the results can be seen in Section 6.6.5. Note that Smurf attacks typically do not pose a significant threat to modern network security anymore. This experiment is meant to simply be an illustrative example with a well-known, third-party dataset. Cybersecurity engineers have the freedom to design encoders however they please to monitor any signal or combination of signals that go beyond scalar values. More about this is discussed in Section 7.2.1.

5.2.4 Measuring Latency

Lastly, we will provide empirical measurements of the increased time complexity of using HOPB compared to the Numenta algorithm in several simulated scenarios. Some increase

in complexity is expected to occur due to the additional computation involved with generating and using HOPB predictions. We use this analysis to additionally motivate a theoretical upper bound for the time complexity associated with using HOPB on top of HTM. It is important to include this analysis to ensure tractable real-time performance is achievable in an actual deployment scenario. A brief asymptotic analysis of using the framework is also presented. The results can be found in Section 6.8.

Chapter 6

Results & Discussion

This chapter will present the results of all the experimentation called for in Chapter 5. The results are discussed along with their presentation and possible explanations are given of each behavior observed.

6.1 A Note on Encoding Time

The values of the timestep increments shown in Figures 6.1-6.3 and 6.19-6.21 do not contain any time of day information and were not used in the encoding process. HTM encoders typically include timestamps into the encodings to more easily learn daily patterns although it is not necessary to do so. In an ideal scenario where the data patterns perfectly repeat according to some constant period, the cyclic encoding of timestamps, if perfectly matched with the periodicity of the signal, will allow the HTM system to learn the relationship between timestamps and data values. In this situation, the system will be more sensitive to certain temporal anomalies such as contextual anomalies because the timestamp combined with the data value illuminates the anomalous pattern. This has been verified in practice by designing the sine wave period to perfectly match up with the passing of 24 hours which the then utilized timestamp encoder was designed to cyclically capture. As long as significant spatial deviation exists between the two contexts, the HTM system uses the timestamp to recognize the contextual anomaly instead of relying on the signal value itself.

This kind of perfect matching can be assumed to almost never occur outside of contrived examples. The encoding of the timestamps would need to be manually adjusted for every pattern in every dataset which is not in line with the goals of a general purpose anomaly detection algorithm. For instance, if a pattern occurs every 10 minutes, then you

would need an encoder that cyclically encodes 10 minute intervals. If the period changes, the encoder would need to be manually adjusted and an entirely new HTM system would need to be trained which destroys HTM's online learning properties. In addition, patterns learned by the HTM system can appear with dynamically long and short periods and often multiple different length patterns exist in the same dataset. This timestamp matching phenomena is impossible in these scenarios. In situations where timestamps don't match up with the signal pattern periodicity, their encoding can actually obfuscate and cloud the signal with useless information. In any case, we would rather have the encoder use its entire representative capacity for the data signal itself. The Numenta algorithm code inside NAB additionally uses one timestamp encoder design for all of the included datasets. This timestamp encoder encodes the time of day, the day of the week, and if the timestamp is on a weekend. However, different datasets from various domains are recorded with vastly different sampling frequencies that might not make sense for this design. In medical data, for instance, the sampling frequency on an electrocardiogram may be fractions of a second and thus it does not make sense to considering what day of the week the electrocardiogram was taken. This limits the potential datasets that can be included inside NAB. Ideally, we want the algorithm to experience the passage of time passively like humans do and use only the changing data signal values to detect all kinds of anomalies in the effort to be more generally applicable and autonomous. For these reasons, *no timestamps are encoded in the experiments of this thesis unless explicitly mentioned.* We test the effect that timestamp encoding has on NAB scores for both the Numenta algorithm and HOPB in Section 6.7.2.1.

6.2 Verification of Motivations

One of the contributions of this thesis is the observation, characterization and attribution of a previously unidentified limitation of streaming anomaly detection algorithms that are restricted to using only first-order predictions. This limitation has been verified through experiments which we will detail and describe in this section.

The principle limitation of the Numenta anomaly detection algorithm in [12] is the limited scope of context used when calculating prediction error. Section 3.1.6.1 describes the limitation in more detail. To verify that this limitation exists beyond the theoretical motivation, several experiments were specially constructed. In each experiment, a simple time-series pattern was constructed (5,000 timesteps of a sine wave normalized to be between 0 and 1 with a constant frequency) with an certain type of anomaly manually placed after sufficient learning of the sequence has occurred. A very simple sequence pattern was

used for the purpose of clear argument. If the limitation is occurring on a very simple sequence, then there is no reason to believe it will not occur on other simple sequences or on any significantly more complex sequences. For these experiments, I used exactly the same underlying HTM model parameters and anomaly detection algorithm implementation as seen in [12] which are openly available in the NuPIC v1.0.3 code repository [120].

All the experiments in this section were repeated with a timestamp encoding introduced that didn't perfectly match up with the periodicity and the same behavior was observed. In fact, the prediction error distributions were shifted upward in general when a timestamp was included in the encoder presumably because of the reasons stated in Section 6.1. Only when the timestamps perfectly coincided with the periodicity of the sine wave was the Numenta algorithm partially able to detect contextual anomalousness. However, the detection was not derived from the data value signal itself (which is desirable for the sake of generality) and it was relatively weak.

6.2.1 Spatial Anomaly Experiment

The first experiment is essentially a sanity check to see that HTM can indeed detect relatively large spatial deviations in time-series data with respect to the natural noise distribution. Evidence is shown in [12] that the anomaly detection algorithm is able to perform well under significant noise do to the statistical modeling procedure on prediction errors. The constructed dataset has no noise, so any spatial deviation at all should be considered anomalous. Either way, a large spatial deviation is manually introduced that spans only three time steps and the results are visualized. The results of this experiment are shown in Figure 6.1.

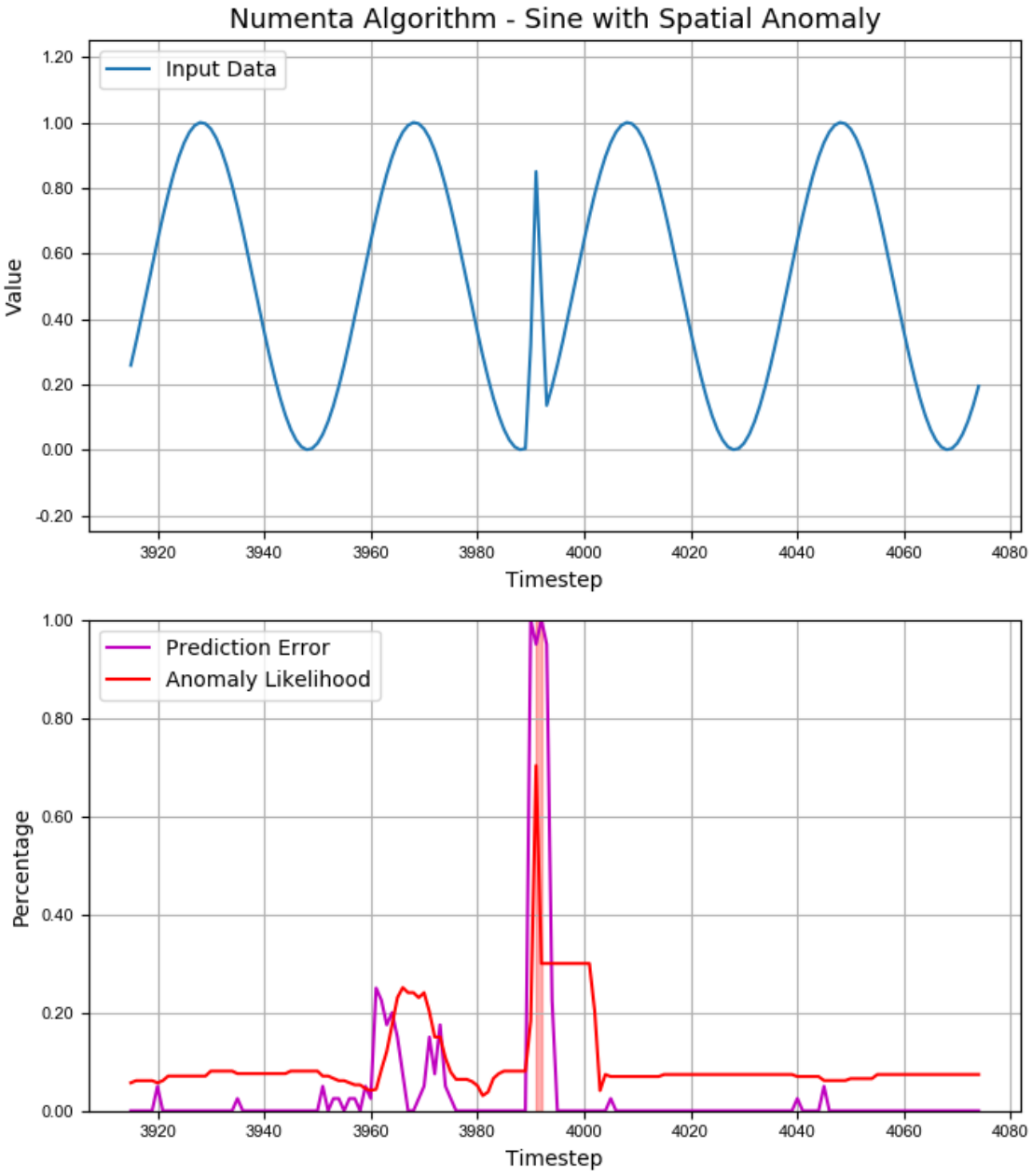

Figure 6.1: *The behavior of the Numenta algorithm for anomaly detection in the event of a large spatial anomaly. The low prediction error (shown in purple) outside of the spatial anomaly indicates that the HTM model has properly learned the sine wave and is predicting it accurately.*

The purple line shows the prediction error which is reflective of the number of overlapping bits between actual activations and predicted activations of columns at each timestep.

The relatively low error outside the anomaly ensures us that the HTM system has learned the sine wave well and is accurately predicting next-timestep network states. The red line details the anomaly likelihood measure which reflects the probability that a given prediction error represents an anomaly based on past prediction errors modeled as a rolling normal distribution [12]. The red highlight indicates that the likelihood measure has breached the threshold which was set at 0.6 and an anomaly has been detected. The exact value of the anomaly likelihood threshold is not important here since we are observing the behavior of the system in general. This experiment simply verified the algorithms were implemented correctly and large spatial deviations are easily detectable as expected since they consist of large first-order transitional spatial deviations.

6.2.2 Contextual Anomaly Experiment

This second experiment included the manual placement of a contextual anomaly in the sine wave to observe the behavior exhibited by the Numenta algorithm. The contextual anomaly was simulated by copying a small piece of the sine wave and placing it where it has never been seen before. This can be interpreted as a sudden and unpredicted context shift which is characteristic of real anomaly models in many application domains. The results can be seen in Figure 6.2.

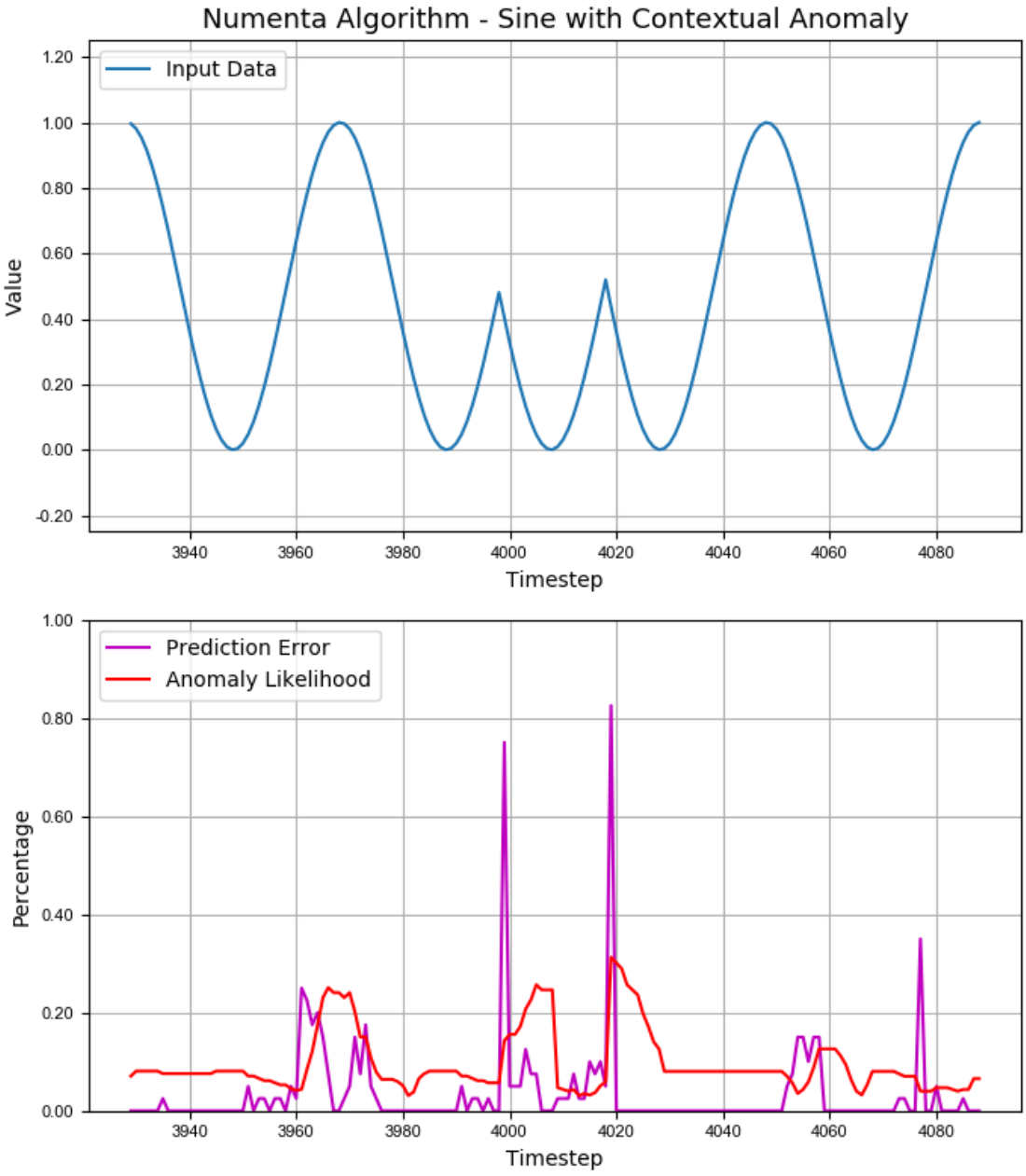

Figure 6.2: *The behavior of the Numenta algorithm for anomaly detection in the event of a contextual anomaly. This experiment illustrates the Numenta algorithm's failure to incorporate high-order sequence context into anomaly detection.*

We see two spikes of error where the contextual anomaly begins and ends because those first-order spatial transitions have not been seen before. However, no significant error at all is reported inside the anomaly because those first-order transitions *have* been

learned before. Once the HTM network passes the initial spatial deviation, it essentially forgets where inside the local sequence context that it is in. It only sees a previously seen data value and predicts the next network state that came immediately after it in the past. This kind of behavior is not desirable for detecting complex contextual anomaly models. The anomaly likelihood probability does not reach a significant value during this event presumably because of the averaging effects over a short-term window of prediction error which are necessary to avoid too many false positives. Even if the short-term averaging feature was turned off, it is clearly undesirable to witness no error being reported inside the anomaly because of the lack of high-order sequence context. Ideally, we want to see prediction error reported inside the anomaly because those data points are anomalous with respect to a high-order context. It is easy to imagine scenarios where no significant transitional spatial deviation would occur at the beginning or end of a contextual anomaly and thus the anomaly would go undetected.

In order to detect this anomaly, we need some sense of high-order sequence context to determine the inner anomaly points as anomalous and thus drive the short-term average of error up. This experiment illustrates the fact that the Numenta algorithm is only able to search for abnormal instantaneous signal transitions to determine anomalous state. In the presence of high-order temporal anomalies, a wider perspective of signal change over time is required. Note that if the sampling frequency on this sine wave were higher, the instantaneous transitional error at the beginning and end of the anomaly would be even smaller and the anomaly likelihood probability would have risen even less.

6.2.3 Collective Anomaly Experiment

The third experiment simulates a collective anomaly in the sine wave. Collective anomalies are loosely defined as a sequence of data points that do not contain enough spatial abnormality in any one data point in isolation to warrant an anomaly. Only when taken together in rapid sequence can they be rightfully considered anomalous. In order to detect these kinds of anomalies, the algorithm needs some sense of accumulating sense of anomalousness over sequential errors. The collective anomaly was simulated by including random spatial deviations in a consecutive sequence of data points. Any single spatial deviation in isolation is relatively minuscule but taken together in rapid sequence should signal an alarm. The results of this experiment are shown in Figure 6.3

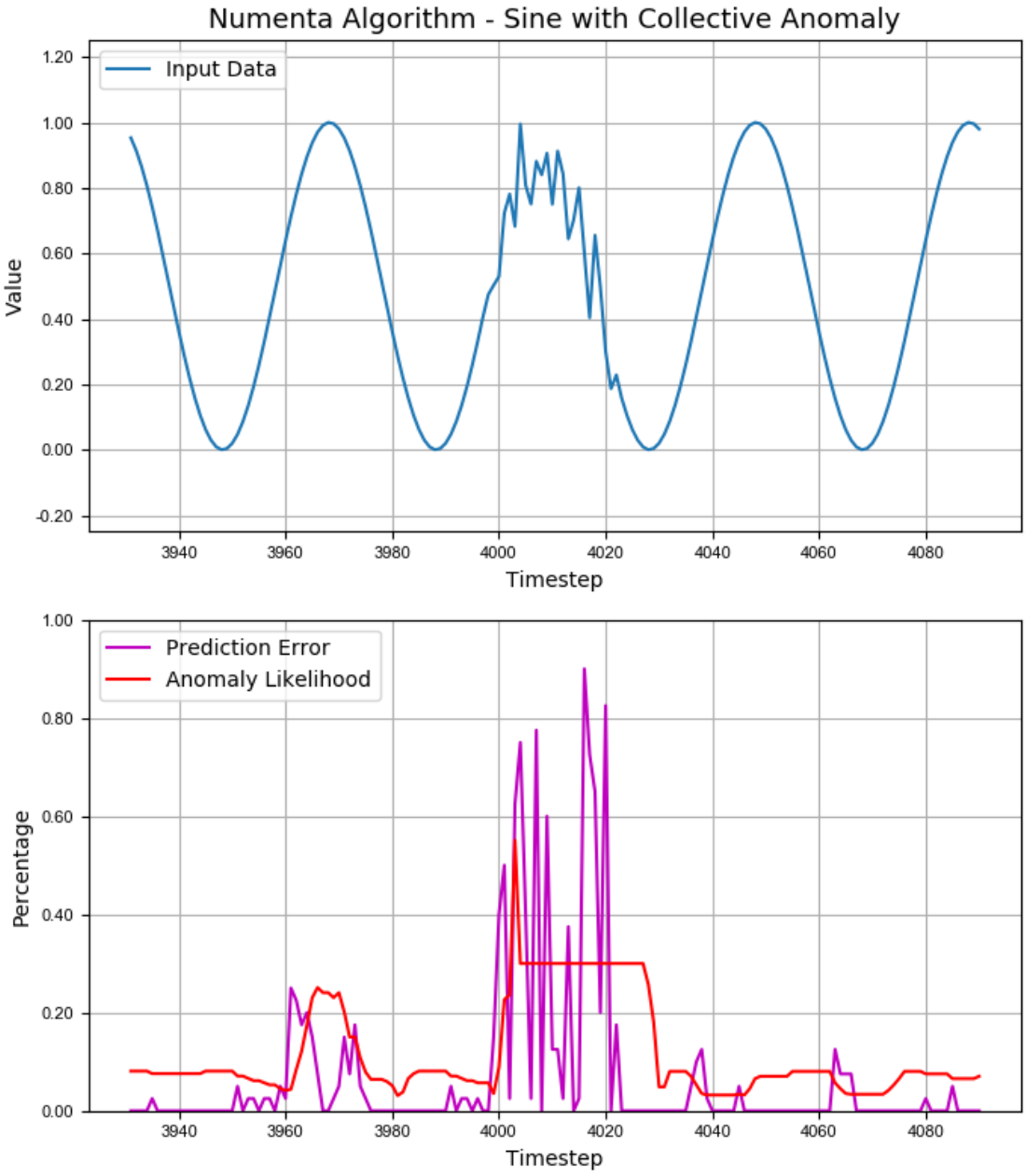

Figure 6.3: *The behavior of the Numenta algorithm for anomaly detection in the event of a collective anomaly. This experiment illustrates the Numenta algorithm's inability to incorporate accumulating broken expectations over time into anomaly detection.*

In this experiment we do see the anomaly likelihood measure rise to approximately 50% because of several recognized spatial deviations in sequence but that response is relatively weak. The prediction error metric for each timestamp is completely independent

of everything before and after it. We would ideally like to see the prediction error rise over time with rapidly repeated broken expectations to incorporate the temporal component of the anomaly in addition to the spatial component. Instead, each small deviation is taken in isolation and their placement in the sequence with respect to each other is not considered despite the fact that that is often an important feature in the signal to detect collective anomaly models. Despite the fact that the prediction error reaches almost 90% at one timestep, the anomaly likelihood probability does not respond much because instantaneous transitional errors have been seen before. Ideally, we want to see a unique increase in error that is reflective of the accumulation of error over a short elapse of time.

6.3 HOPB Verification Experiments

It is necessary to test the efficacy of HOPB predictions themselves with respect to their ability to solve the problem defined in Section 1.3 and illustrated in Section 6.2. Additionally, empirical justifications for the limiting factors discussed in Section 4.3.1 are provided. Two sets of experiments were set up for this purpose. The first set of experiments measures the density and prediction error separately for each order prediction across an entire data stream for the sake of comparison and observation of behavior in various scenarios. The second set of experiments explicitly tests the ability of HOPB predictions to capture sequence context information.

6.3.1 Density Experiments

It has been experimentally observed in physical brains that overall cell activity in the cerebral cortex becomes sparser during a continuous predictable sensory stream [80, 123, 128]. HTM theory provides an explanation for this phenomenon by design. When using HOPB predictions, the density of the HOPB prediction SDR is of key importance in the determination of its reliability. If the density of a HOPB prediction SDR is too high or too low, the prediction errors will either erroneously plunge or skyrocket accordingly. When it comes to anomaly detection, consistency is key.

To test the effects of density, four artificial datasets were constructed. The first dataset consists of randomly generated points that do not follow any discernible pattern. This is in an attempt to capture the case maximum uncertainty where we would expect the maximum amount of possible futures to be predicted and thus the largest increase in SDR density. The second dataset is a simple unaltered sine wave. This is in an attempt to capture the case of high predictability where we would expect the minimal amount of possible futures to be predicted and thus a minimal increase in SDR density. The

third and fourth datasets represent sine datasets with varying levels of Gaussian noise. The third dataset introduces relatively small noise with a maximum amplitude of 1% of the total range and the fourth dataset features relatively large noise that is 5 times the amplitude of the third dataset.

Each dataset consists of 5000 timesteps. Sample prediction SDRs are only collected after the network has sufficiently learned the signal. This is ensured by a 15% probationary period. Statistics are calculated on the products of HOPB with this data both with and without adding density constraints on the HOPB prediction SDRs. Recall the density of an SDR is measured by the number of active bits over the total number of bits.

6.3.1.1 Using Random Data

The results of using completely random data are displayed in Figures 6.4-6.5.

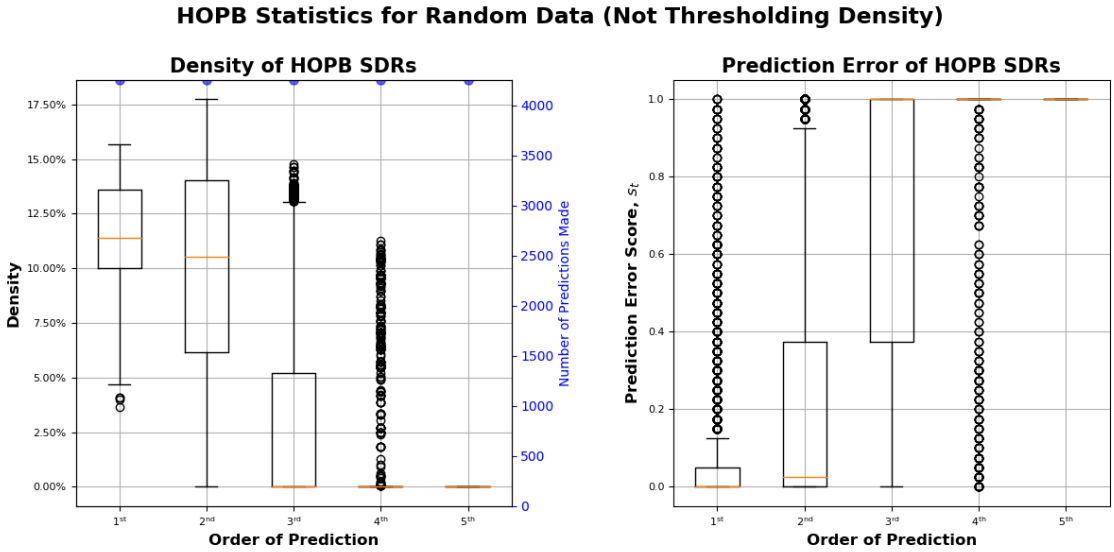

Figure 6.4: *HOPB prediction statistics gathered while not thresholding density on randomly generated data.*

In Figure 6.4, we see that not having any density constraints results in *many* failures to generate non-empty fourth-order predictions and no successful attempts to generate non-empty fifth-order predictions. These empty predictions are due to the network not possessing the lateral connections necessary to move this far out in time. Clearly, these empty predictions would present an enormous hurdle to reliable performance and not thresholding density cannot be expected to work.

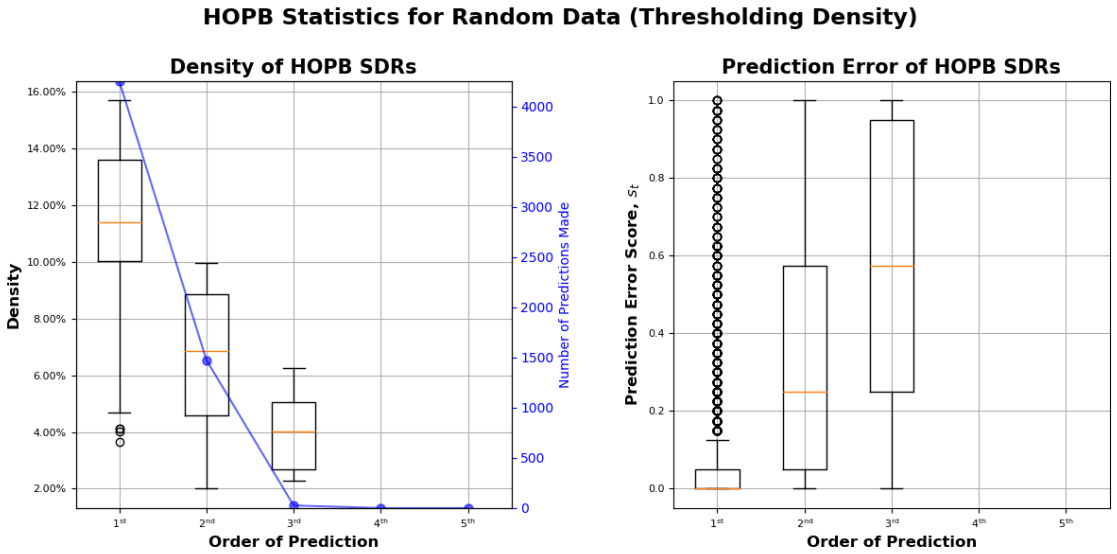

Figure 6.5: *HOPB prediction statistics gathered while thresholding density on randomly generated data.*

In Figure 6.5, we present a threshold of acceptable density between 2% and 10% for high orders of prediction. If an HOPB prediction SDR comes along that does not meet the density requirement, it is thrown away and no further predictions are made. We can see the characteristic effects that this threshold has on error and the number of predictions made of each order. Concretely, when density constraints are imposed:

1. The trend of prediction errors of each order will decrease.

2. The number of predictions made at each order will decrease.

These effects are amplified the further out you go in time. In essence, we are simply discarding predictions that we know have no chance of being reliable due to incorrect density.

In both of these experiments, notice how the average density of even first-order predictions is very high. This is a property of the temporal memory algorithm by design. When presented with an unpredictable stream of information, bursting is abundant within the network and many columns are predicted in general. The fact that so many columns are being predicted explains why we see relatively low prediction error even though the dataset is completely unpredictable. We will not see this high of density in datasets that follow some predictable pattern of some sort. 10% as a maximum density threshold is a better option for general purpose applications.

6.3.1.2 Using a Perfect Sine Wave

The results of using a perfect sine wave are displayed in Figures 6.6-6.7.

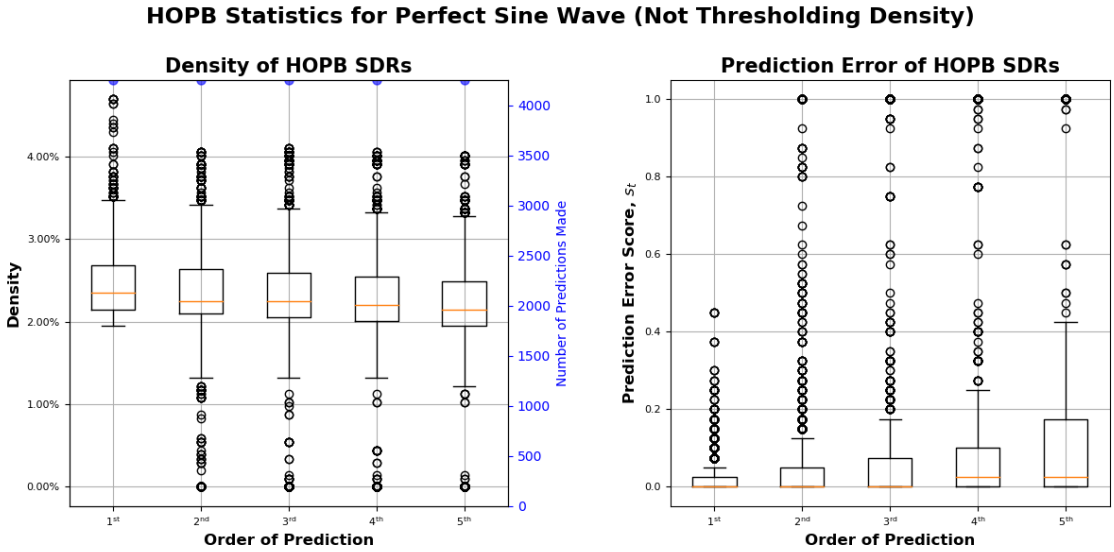

Figure 6.6: *HOPB prediction statistics gathered while not thresholding density on a perfect sine wave.*

Notice in Figure 6.6 a density range that falls very tightly capped just above 5%. This is because the sine wave is perfectly predictable and the level of uncertainly in the network is low. However, notice that even in this scenario, we see many random plunges of density down below 2% when using high-order predictions. This is because accurate high-order predictions demand a complex architecture of lateral connections to be established which is in general more difficult to obtain (takes longer) than accurate first-order transitions. These random plunges of density are undesirable, as seen by the random spikes of prediction error that they result in in high orders of prediction seen on the right.

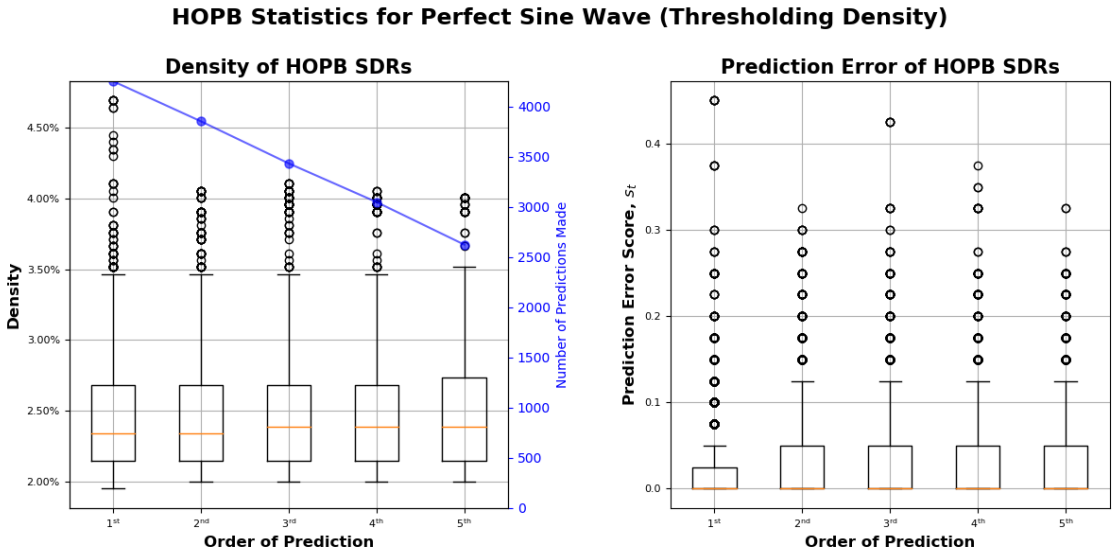

Figure 6.7: *HOPB prediction statistics gathered while not thresholding density on a perfect sine wave.*

Alternatively, by threshold density to be between 2% and 10%, we see a total disappearance of these random spikes of error. Also, the number of predictions does not suffer incredibly as we see by fifth-order predictions we're still able to produce just above 2500 acceptable predictions out of the maximal 4250. The distribution of prediction error of high orders of prediction are relatively the same as first-order which is desirable. Note the subtle homogeneity of the distribution of prediction errors among high orders on the right. Even though the density distribution errors vary of these predictions, it is hypothesized that this homogeneity is due to the network learning a single sequence of transitions that represent the sine wave that cycle around. Thus, that same cycle is carried out in some capacity at each timestep and order of prediction and the overlapping chains result in similar error distributions at each order.

6.3.1.3 Using Sine Wave with Gaussian Noise

We additionally want to analyze the density characteristics of sine waves that are perturbed with Gaussian noise. We introduce Gaussian noise with 5% of the total data range in amplitude into the sine wave. The results of this experiment are displayed in Figures 6.8-6.9.

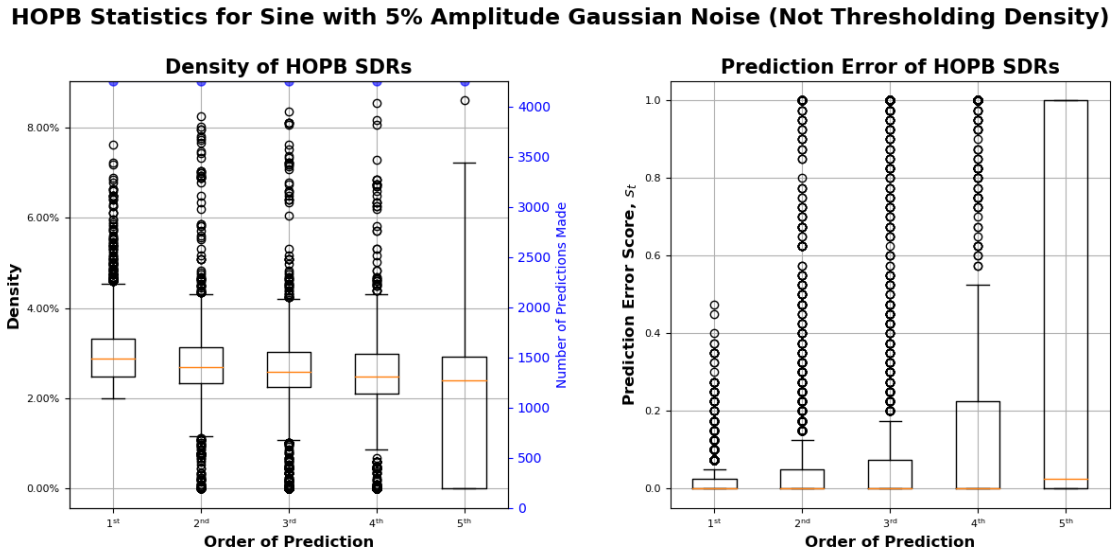

Figure 6.8: *HOPB prediction statistics gathered while not thresholding density on a sine wave perturbed with Gaussian noise with amplitude equal to 5% of the total range of values.*

In Figure 6.8, we see the number of predictions with unacceptable density to be very plentiful in high orders of prediction. The effects that this has on prediction error scores are much more devastating than when using a perfect sine wave as expected. The decrease in performance at which increasing orders of prediction seem to behave appears to be nonlinear. In Figure 6.9 we threshold the density of SDRs to be between 2% and 10%.

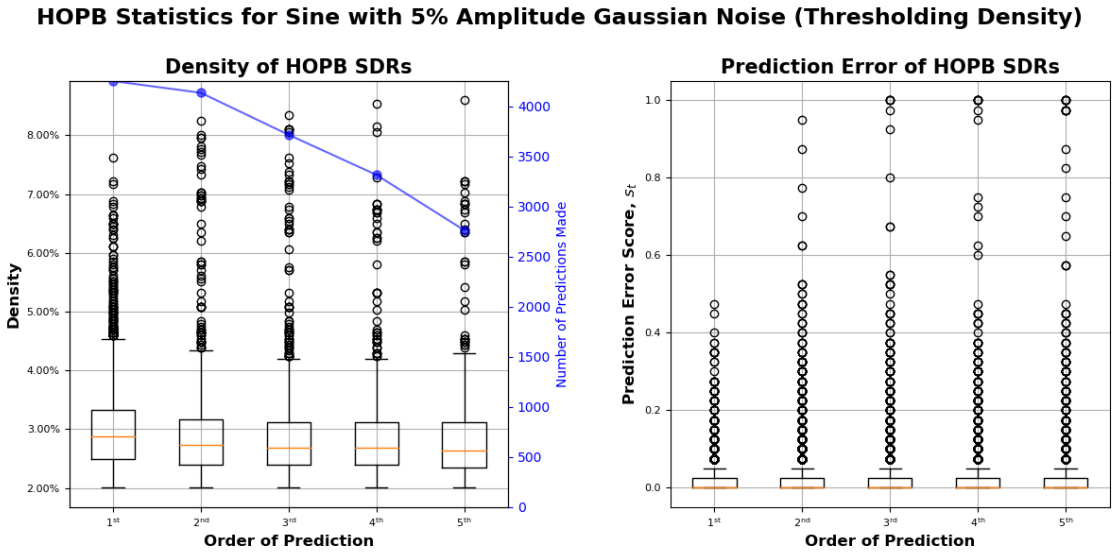

Figure 6.9: *HOPB prediction statistics gathered while thresholding density on a sine wave perturbed with Gaussian noise with amplitude equal to 5% of the total range of values.*

It is important to note the results obtained here in Figure 6.9. We see the same effects that thresholding density typically has, but we see that thresholding density alone is not enough to ensure the reliability of a high order of prediction. Even with all the SDRs of an order possessing acceptable density, we still have many erroneous bursts of error at high orders of prediction due to the shear unpredictability of the data at high orders. Thus, in order to measure the reliability of an order of prediction accurately, we need to incorporate some sense of the order of prediction's past prediction error performance. Ideally, only those orders of prediction that possess a similar prediction error distribution as first-order predictions should be used.

6.3.1.4 A Hidden Potential

The effect of thresholding SDRs on density is dramatic. What we often see for a high order of prediction is the presence of reasonable density SDRs hidden within the total population of SDRs. For a visualization of this, see Figures 6.10 and 6.11.

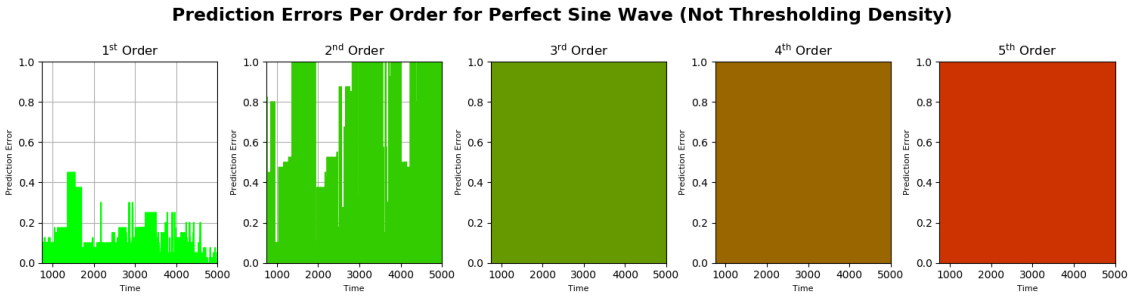

Figure 6.10: *HOPB prediction errors per order during the processing of a sine wave dataset while not thresholding on density. Note the maximally high error in the high-order predictions.*

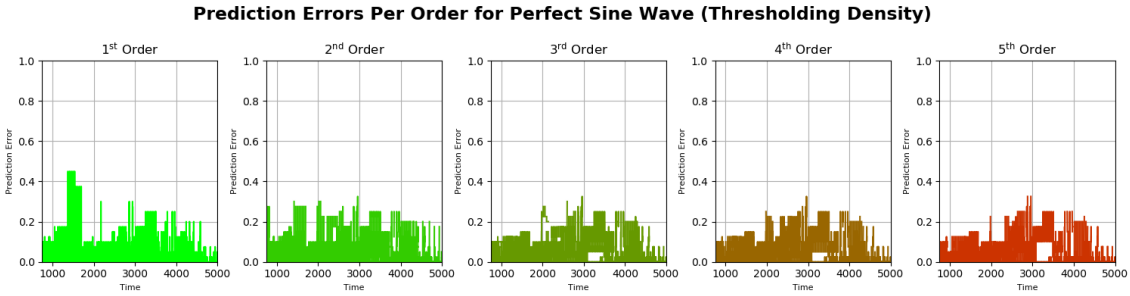

Figure 6.11: *HOPB prediction errors per order during the processing of a sine wave dataset while thresholding on density. Note the existence of many reasonable density SDRs with low error at all orders of prediction.*

In Figure 6.10, we see that the high prediction error in high orders of prediction is dominating any low error predictions that come through. When comparing it to Figure 6.11, we see that by simply thresholding the density we obtain many predictions that have as low error as first-order predictions. Thus, in order to get a sense of the true value of an order of prediction, we can only judge it by the HOPB prediction SDRs it produces that are of an acceptable density. If we punish an order of prediction based on the HOPB prediction SDRs it produces that are of erroneous density, we will often greatly underestimate the true value of keeping that order of prediction included. That would discard a lot of potential contextual information that can be used for intelligent anomaly detection. Thus, if an order of prediction produces any HOPB prediction SDRs for a timestep that are not of reasonable density, we shouldn't bother ever computing it's prediction error and instead simply make note of a failure for that timestep in order to track the frequency of acceptable predictions. Recall in this thesis we define acceptable SDR density to be between 2% and 10%. See Section 4.3.1 for an explanation why those specific numbers have been chosen.

6.3.2 Chain of Context Experiments

This experiment was intended to investigate if HOPB predictions have the capacity to capture the kind of contextual information that is needed to illuminate an anomalous context shift better than first-order transitional error. For this experiment, we captured and analyzed the HOPB prediction SDRs made at one instant of time while thresholding on SDR density. The instant of time in which we capture the chain is strategically placed just before an anomalous context shift occurs on a sine wave. The same procedure is repeated during the same instant of time on a copy of the same sine wave that does not include a contextual anomaly for comparison purposes. We want to compare how the predicted internal state looks like with respect to normal and contextually anomalous

data separately. This experiment is repeated when the pattern is perturbed with various levels of Gaussian noise to analyze the stability of HOPB predictions.

Note that by "with respect to" a dataset in the following plots of these experiments, we mean with respect to the spatial pooler activations that occur at that time with that data value. Before the context shift, the network is initialized the same exact way and learns the same way from the same data. Thus, before the context shift, the network states are exactly equal. The HOPB prediction SDRs are thus the same at the instant of time before the context shift. Those HOPB prediction SDRs are compared to the SDRs that the spatial pooler produces on each dataset separately, which are naturally different. The data values themselves do not play any direct role in the comparison other than they are what prompt the spatial pooler to produce different SDRs of activated columns. The results when using a perfect sine wave pattern are shown in Figure 6.12.

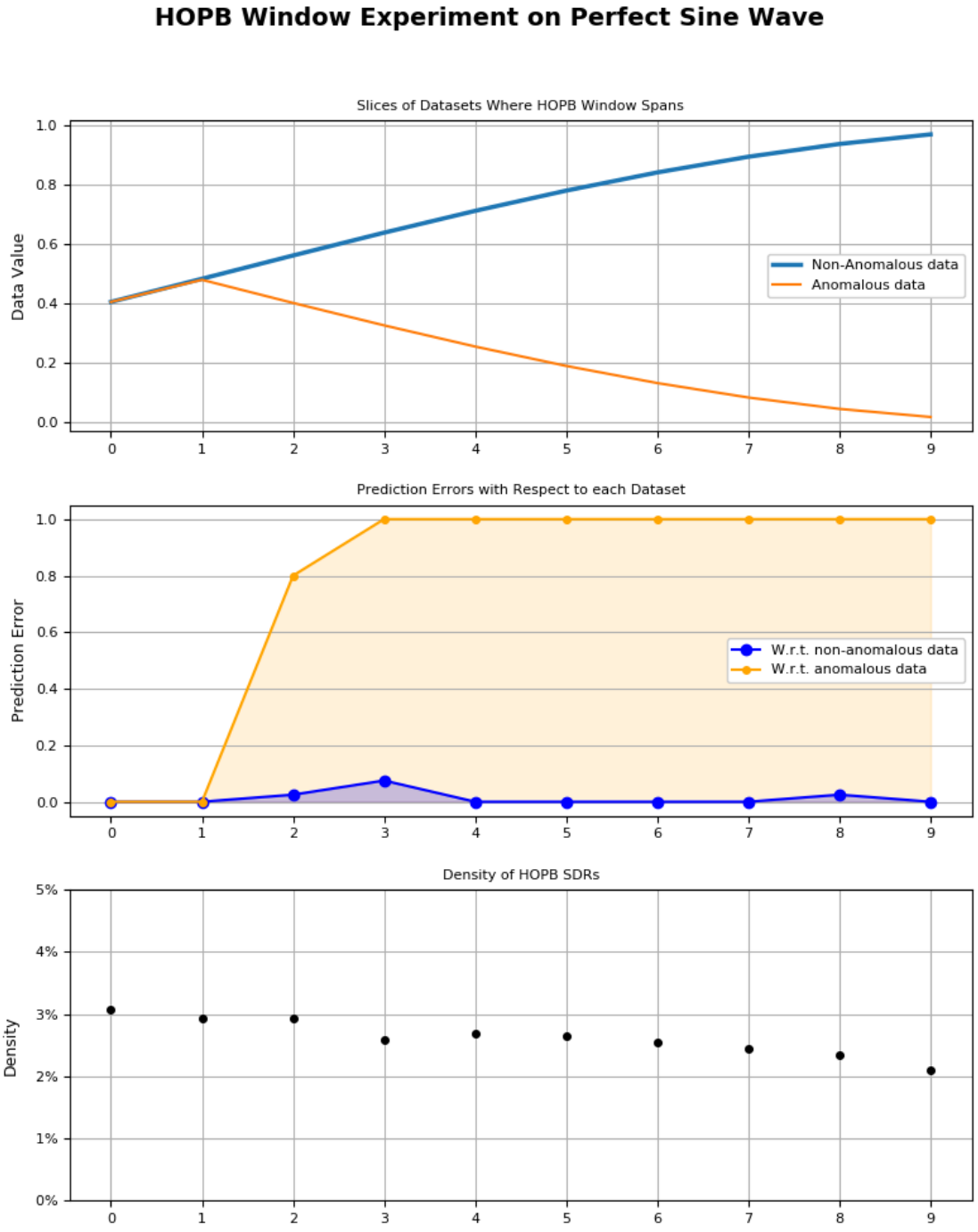

Figure 6.12: *An in-depth comparative look at a chain of HOPB predictions during an anomalous context shift of a perfect sine wave pattern with respect to the anomalous data and what is expected.*

We see in Figure 6.12 that the HOPB predictions are essentially able to perfectly capture the contextual difference 10 timesteps into the future. The prediction error of the HOPB prediction SDRs with respect to the contextually anomalous data goes to maximal very quickly when the context shift takes place. We also see the HOPB prediction SDR density remain relatively constant which is important for checking the validity of the results. However, perfect signals are not realistic. It is instructive to repeat this test under noisy conditions.

Figure 6.13 shows the behavior when the sine wave is perturbed with Gaussian noise that has an amplitude of 5% of the total range of data values.

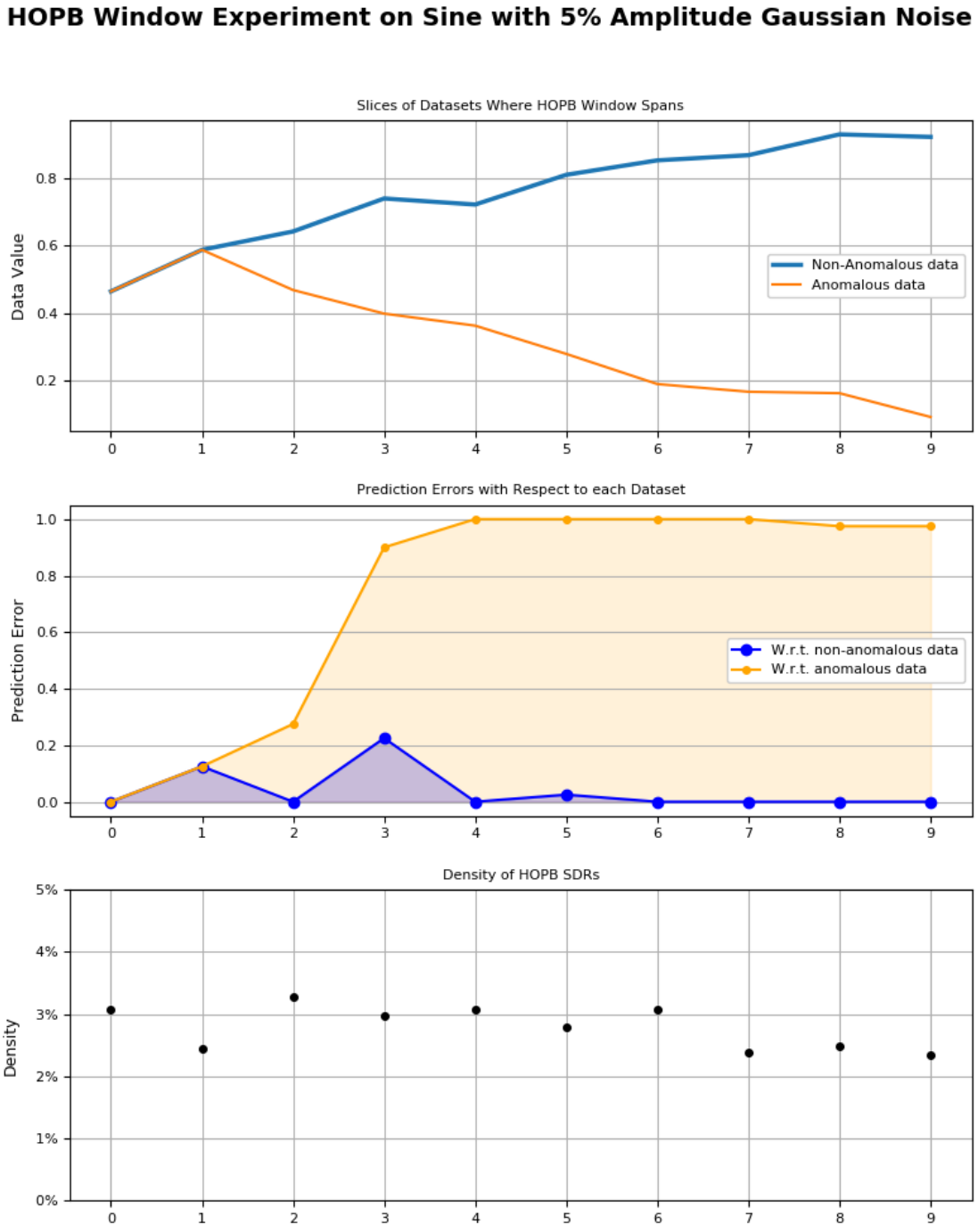

Figure 6.13: *An in-depth comparative look at a chain of HOPB predictions during an anomalous context shift of a sine wave pattern perturbed with 5% amplitude Gaussian noise with respect to the anomalous data and what is expected.*

Figure 6.13 tells a similar story with respect to the ability of HOPB predictions to reliably predict the high-order future this time under noisy conditions. Note that the shift upwards in prediction error with respect to the anomalous data is slightly delayed assumably because the network's predictions are less precise in general due to noise. It takes slightly longer to detect the contextual difference. However, we do manage to see higher error with respect to the anomalous data at all points after the sudden context shift. The prediction errors with respect to the anomalous data do eventually reach maximal as the two streams diverge. The density similarly remains stable, also.

The amplitude of the noise was doubled to 10% for the last experiment of this kind. The results are shown in Figure 6.14.

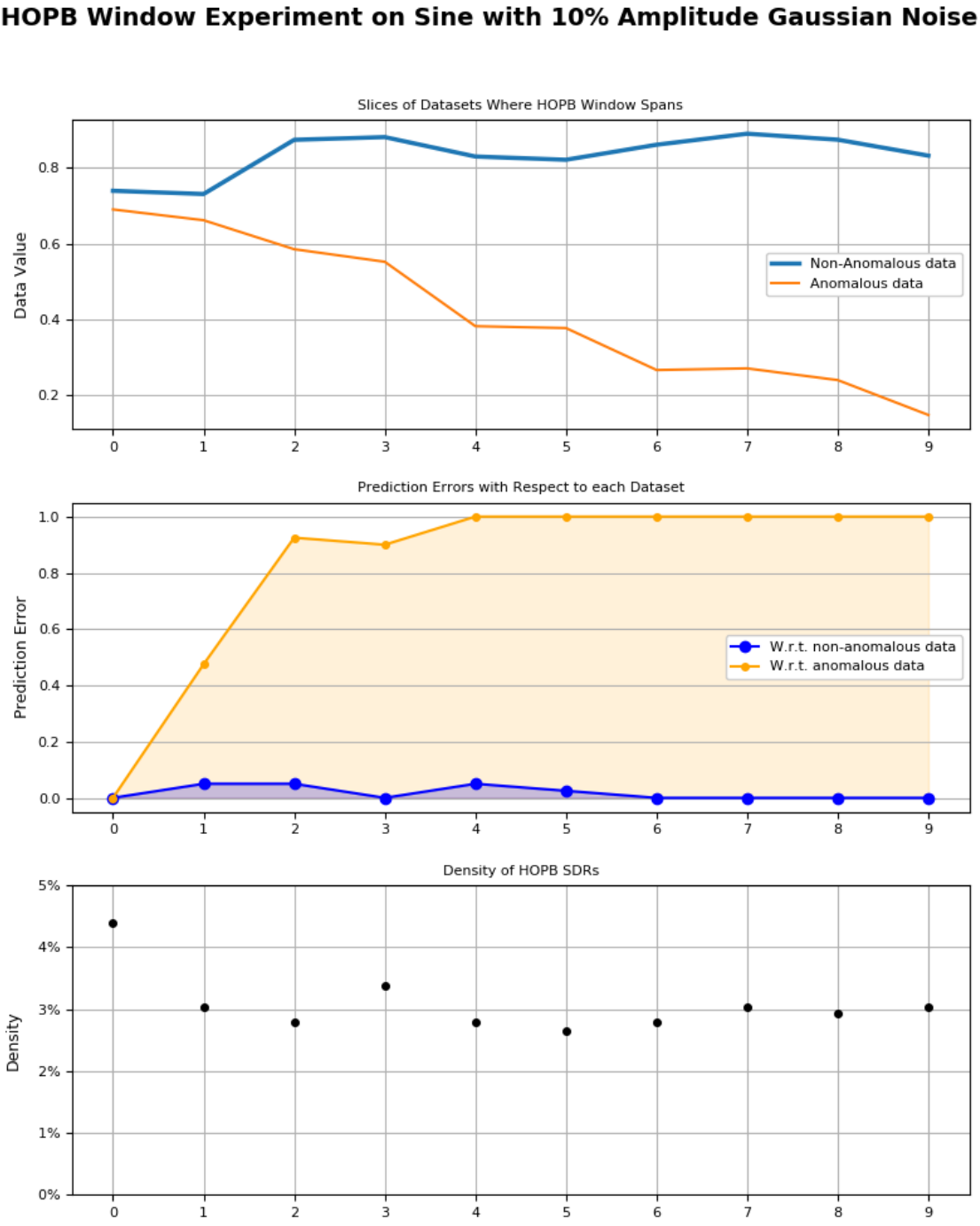

Figure 6.14: *An in-depth comparative look at a chain of HOPB predictions during an anomalous context shift of a sine wave pattern perturbed with 10% amplitude Gaussian noise with respect to the anomalous data and what is expected.*

Interestingly, Figure 6.14 shows that the usefulness of HOPB predictions is robust to relatively high levels of noise. Note that not all HOPB chains are going to be this useful. These experiments simply provide evidence that high-order predictions are indeed capable of capturing useful information.

6.4 Dynamic Chain Size Experiments

This section is intended to explore the behavior of using a dynamic chain size with HOPB as described in Section 4.3.3. Recall the goal of using a dynamic chain size is to autonomously discover the optimum number of predictions to make per timestep given the current state of signal characteristics and the underlying HTM model. Both are subject to constant change in a real-time stream of data and we need to engineer HOPB to automatically adapt to ensure stable performance.

6.4.1 Searching for Convergence

Recall in Section 4.3.2 the argument for why a consistent number of predictions is desirable. This translates to a value of n_t that converges over time. Convergence of n_t is largely due to the main tunable parameters of HOPB, ψ, α and ϵ. See Figure 6.15 for an example of reasonable parameter settings that result in the convergence of n_t at approximately four orders of prediction. Small temporary fluctuations to five orders of prediction are due to the nature of Algorithm **??** always trying to experiment with a new order of prediction.

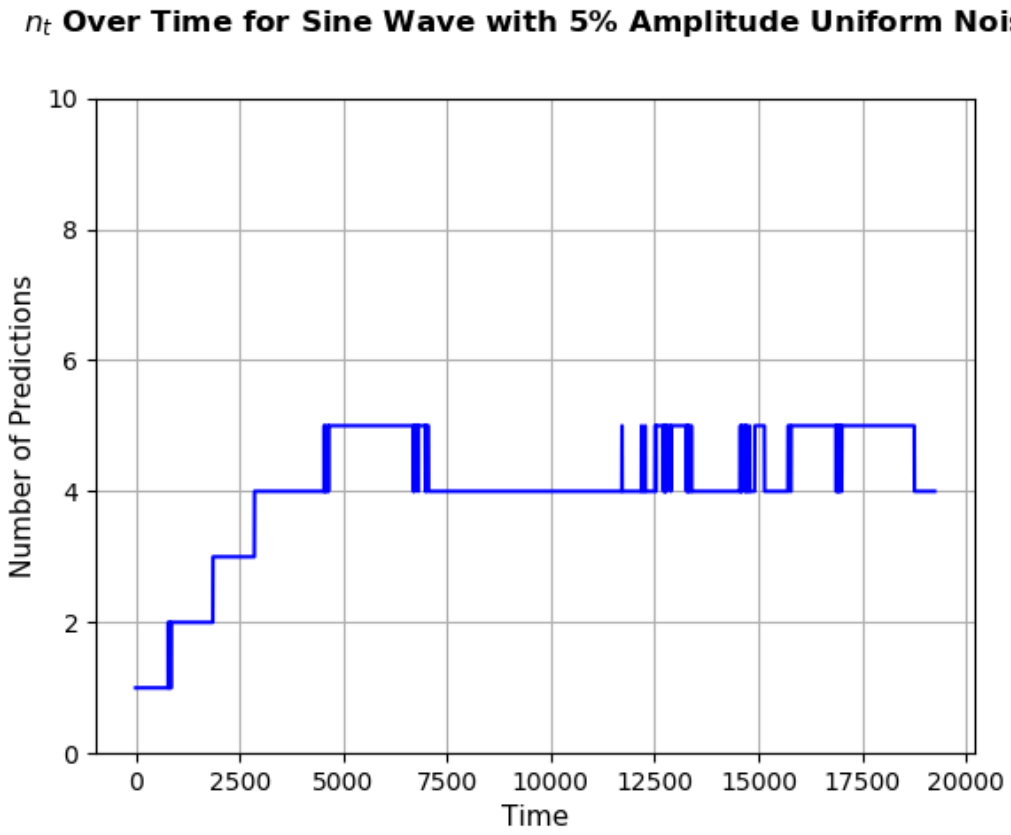

Figure 6.15: *Convergence of n_t during a constant pattern due to reasonable parameter settings. In this case, $\psi = 1000$, $\alpha = 0.8$ and $\epsilon = 0.03$.*

In Figure 6.16, we see what happens if we use a value of ψ that is too small. We reduce the size of ψ from 1000 to 100. Using a small value of ψ means we are sampling the statistical properties of an order of prediction from a very small sample. What results is instability. The number of predictions called for varies widely as each small sample fails to capture the true global characteristics of the order of prediction for that time. If this is happening, it is recommended to increase the value of ψ.

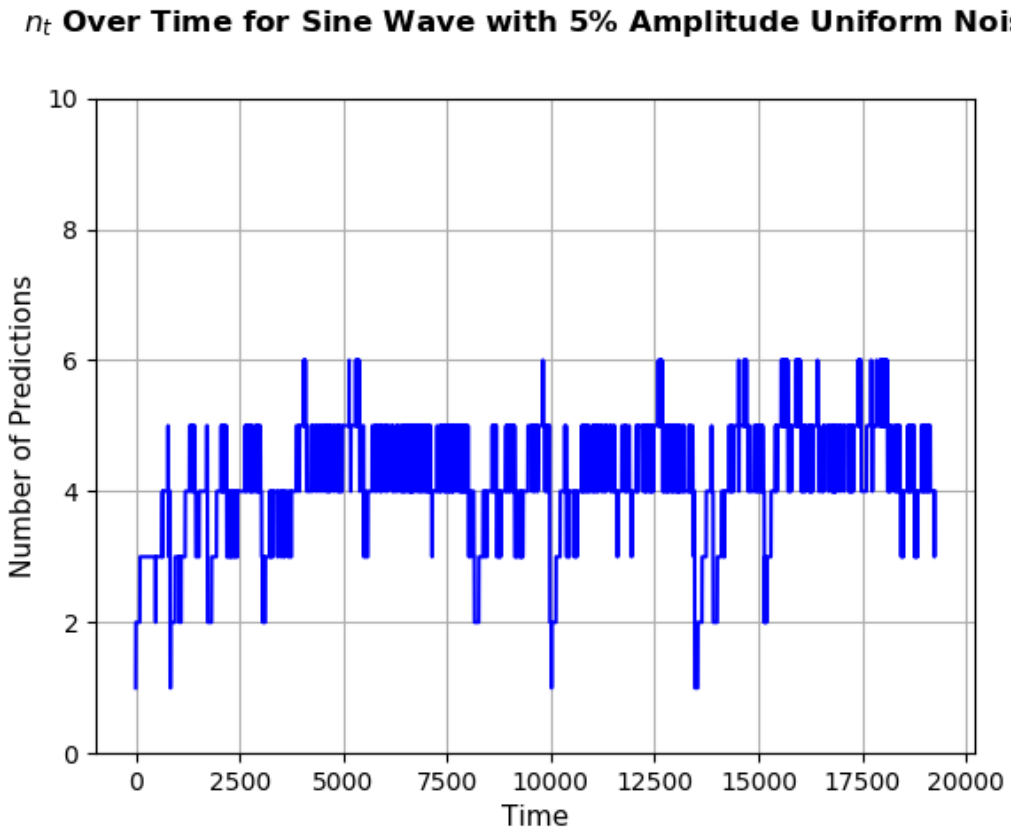

Figure 6.16: *Visualization of the effects on n_t of using too small of a value for ψ. In this case, we reduced the value of ψ from 1000 to 100 from the settings in Figure 6.15.*

In Figure 6.17, we see what happens if we use a value of ψ that is too small *combined with* poor values of α and ϵ. We keep the value of ψ at 100 but reduce the value of α from 0.8 to 0.5 and increase the value of ϵ from 0.03 to 0.1. Values for α that are too low will incorporate too many orders of prediction too quickly. Using a value of ϵ that is too high has a similar effect. What results is the same quickly varying fluctuations seen in Figure 6.16 but, in this case, the fluctuations are more dramatic. The fluctuations reach higher maximums and lower minimums. If this is occurring, it is sign that the value of ϵ is too large or the value of α is too low.

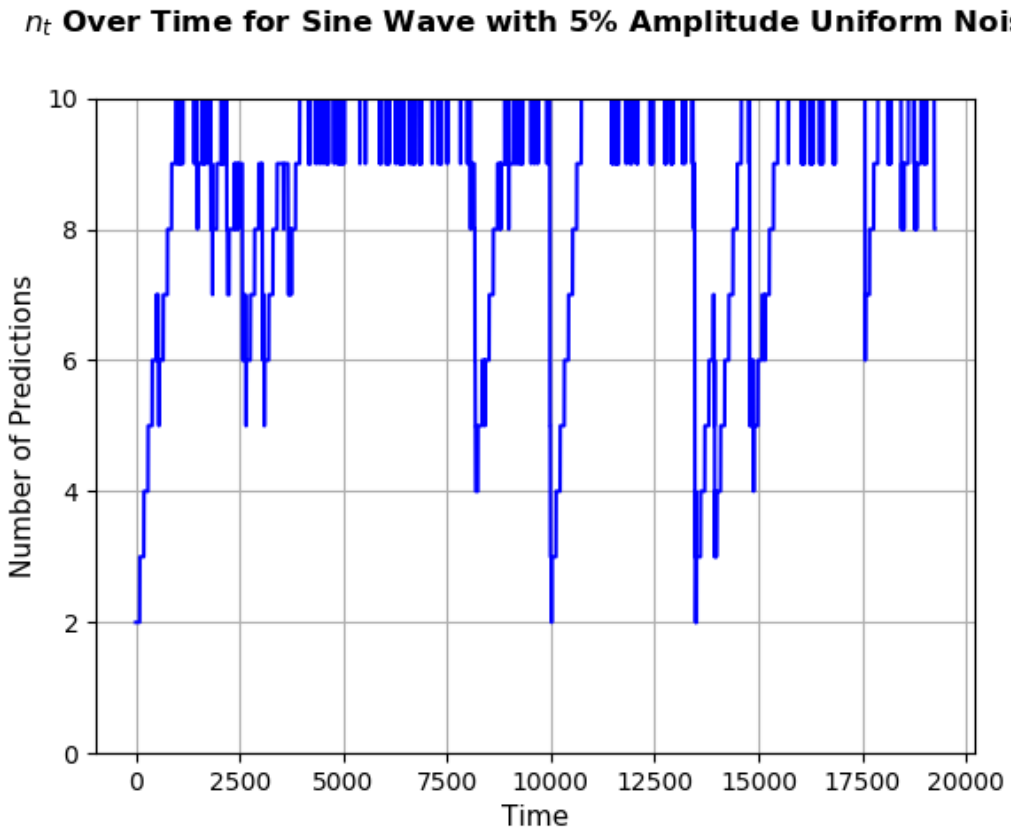

Figure 6.17: *Visualization of the effects on n_t of using too small of a value for α or too large of a value for ϵ. In this case, we reduced the value of α from 0.8 to 0.5 and increased the value of ϵ from 0.03 to 0.1 from the settings in Figure 6.16.*

6.4.2 Visualization of the Benefit

It is important to vary the number of predictions based on what is currently appropriate. Somewhat also demonstrated in Section 6.3.1.3, not all predictions that have a reasonable density are still useful. This can happen if it is simply not possible to predict a signal out to a certain point in time due to uncertainty. See Figure 6.18 to visualize this directly.

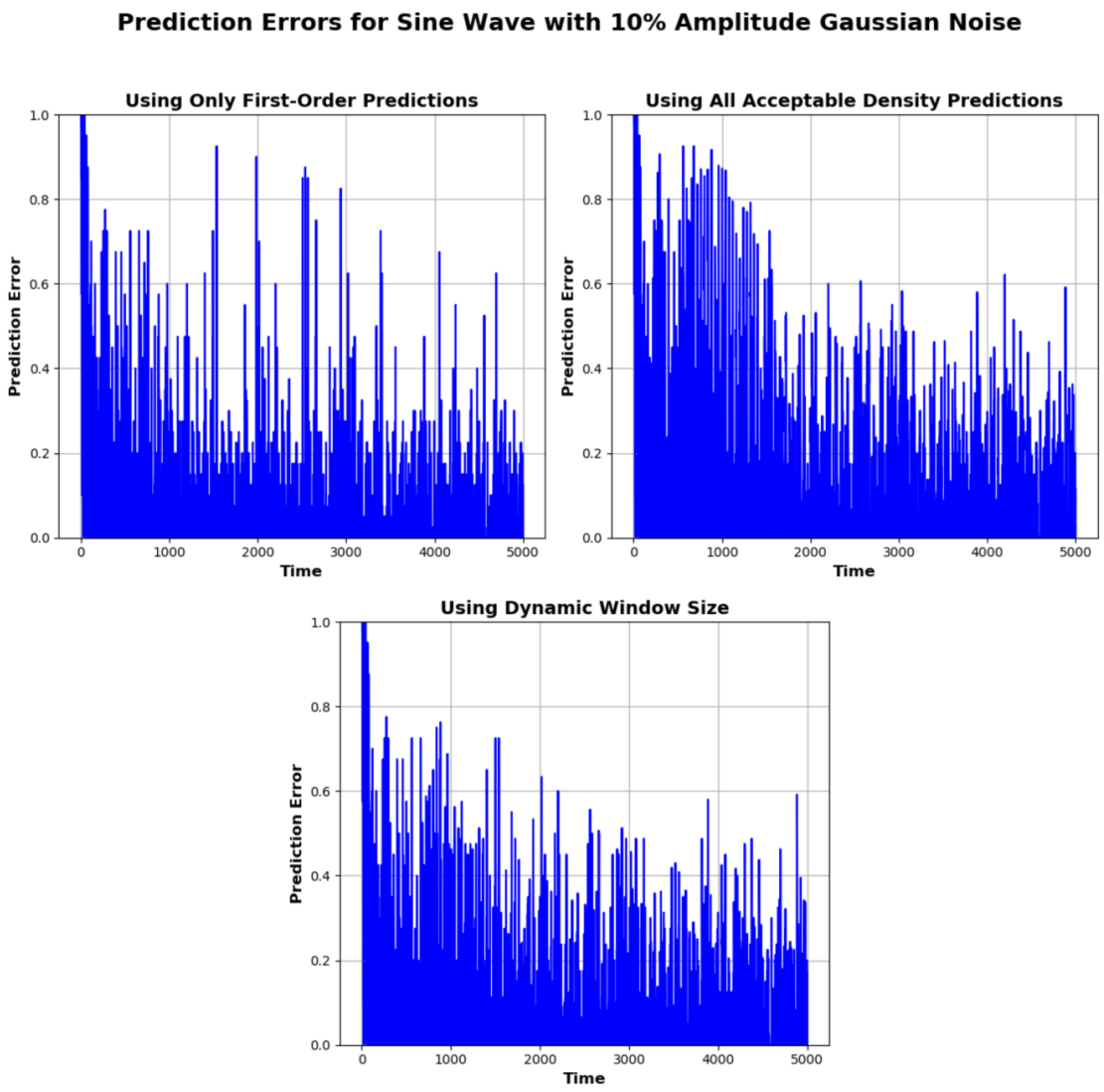

Figure 6.18: *Visualization of how using a dynamic chain size can reduce erroneous high prediction error.*

In Figure 6.18, we see the prediction error over time for a noisy sine wave from the very beginning before any probationary period. Especially in the early stages before sufficient learning takes place, we see using a fixed number of predictions (ten, in this case), can severely hinder performance of the algorithm even while thresholding on density. In the lower plot, when using a dynamic chain size, we see high-order predictions are not used until they provide a reasonable benefit and provide for more consistent performance.

Note that the situation inside Figure 6.18 does not always just happen at the beginning of a new dataset. The beginning of a new dataset simply capitalizes on the idea

of the network not possessing the proper lateral connections for high-order predictions. This situation can also occur if the characteristics of the underlying data change and old transitions are not useful anymore. By virtue of the algorithm, the number of predictions would automatically adjust the same way as it does in 6.18.

6.5 HOPB Behavior Experiments

In this section, we explore the resulting behavior of using these HOPB predictions for the purpose of anomaly detection. For these experiments, we use the full HOPB framework including the dynamic chain size and accumulating average. General purpose HOPB parameters are used in the NAB experiments in Section 6.7. For the following three experiments, we slightly tweak those numbers to somewhat aggressively allow for high-orders of prediction in order to show their benefit. The exact parameters used for all three experiments in this section are $n_{\max} = 10$, $\psi = 500$, $\alpha = 0.5$ and $\epsilon = 0.1$.

6.5.1 In the Presence of a Spatial Anomaly

We present the HOPB algorithm with the same spatial anomaly model as seen in Section 6.2.1. The results can be seen in Figure 6.19. Note that the anomaly model is still immediately recognized even though it consists of only one timestep. The accumulative average reduces the reported prediction error, as expected. However, the relative difference in prediction error illuminated by all orders of prediction agreeing on this timestep which was improbable to guess is still strong enough to bring the anomaly likelihood probability up.

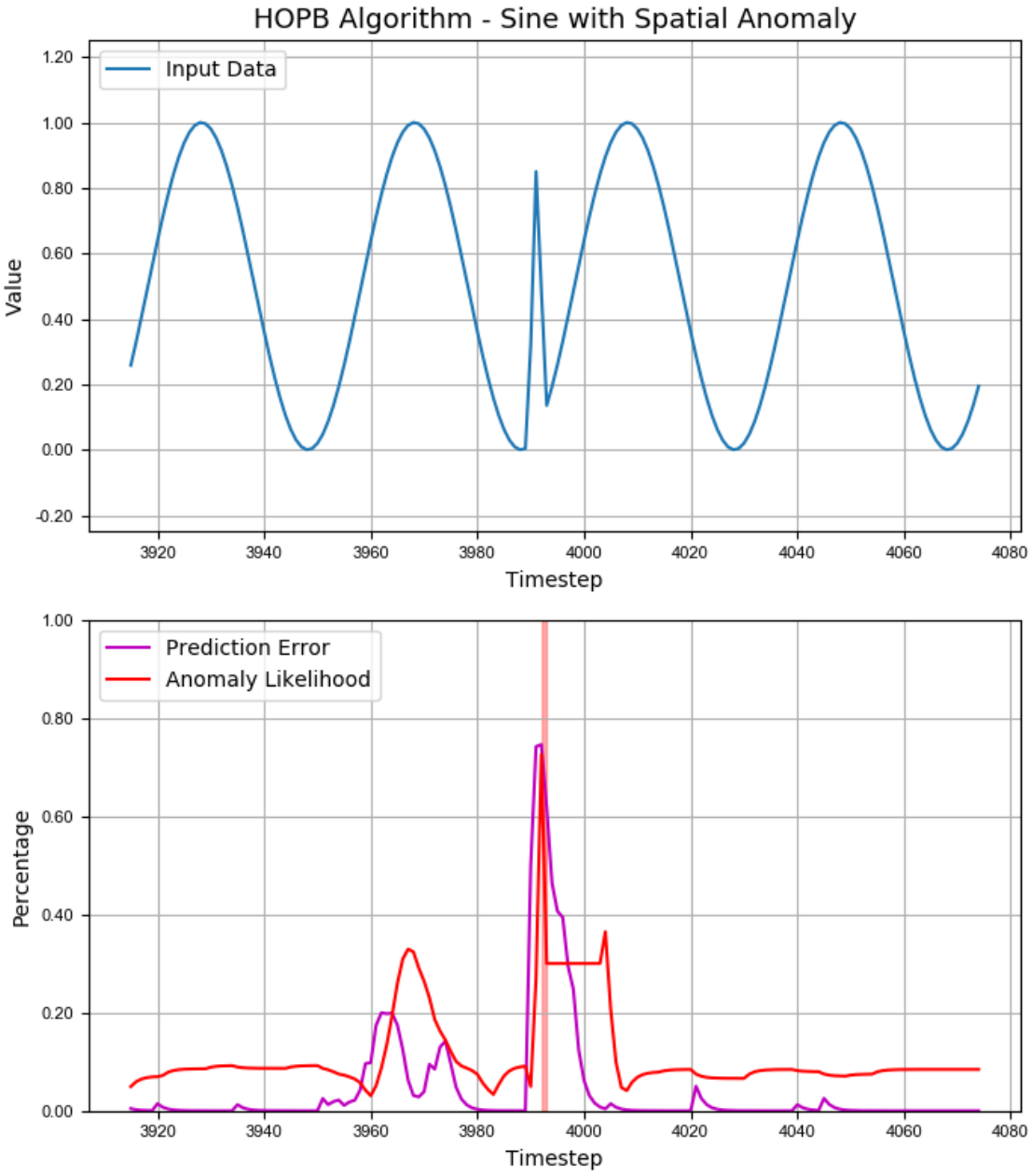

Figure 6.19: *Visualization of the HOPB algorithm's behavior in the presence of a spatial anomaly. Note the ability of HOPB to recognize the anomaly even though the anomaly consists of only a single timestep. Recall the red highlight indicates that an anomaly has been identified by the anomaly likelihood breaching the threshold probability.*

6.5.2 In the Presence of a Contextual Anomaly

We present the HOPB algorithm with the same contextual anomaly model as seen in Section 6.2.2. The results can be seen in Figure 6.20. Note that this anomaly model contains two context shifts at the beginning and end of the anomaly model. As intended, we see multiple timesteps immediately after each context shift experience high error as the contextual anomalousness is being captured. By virtue of the dynamic chain size algorithm, the number of predictions is not reduced after the first context shift because all orders of prediction experienced similar error.

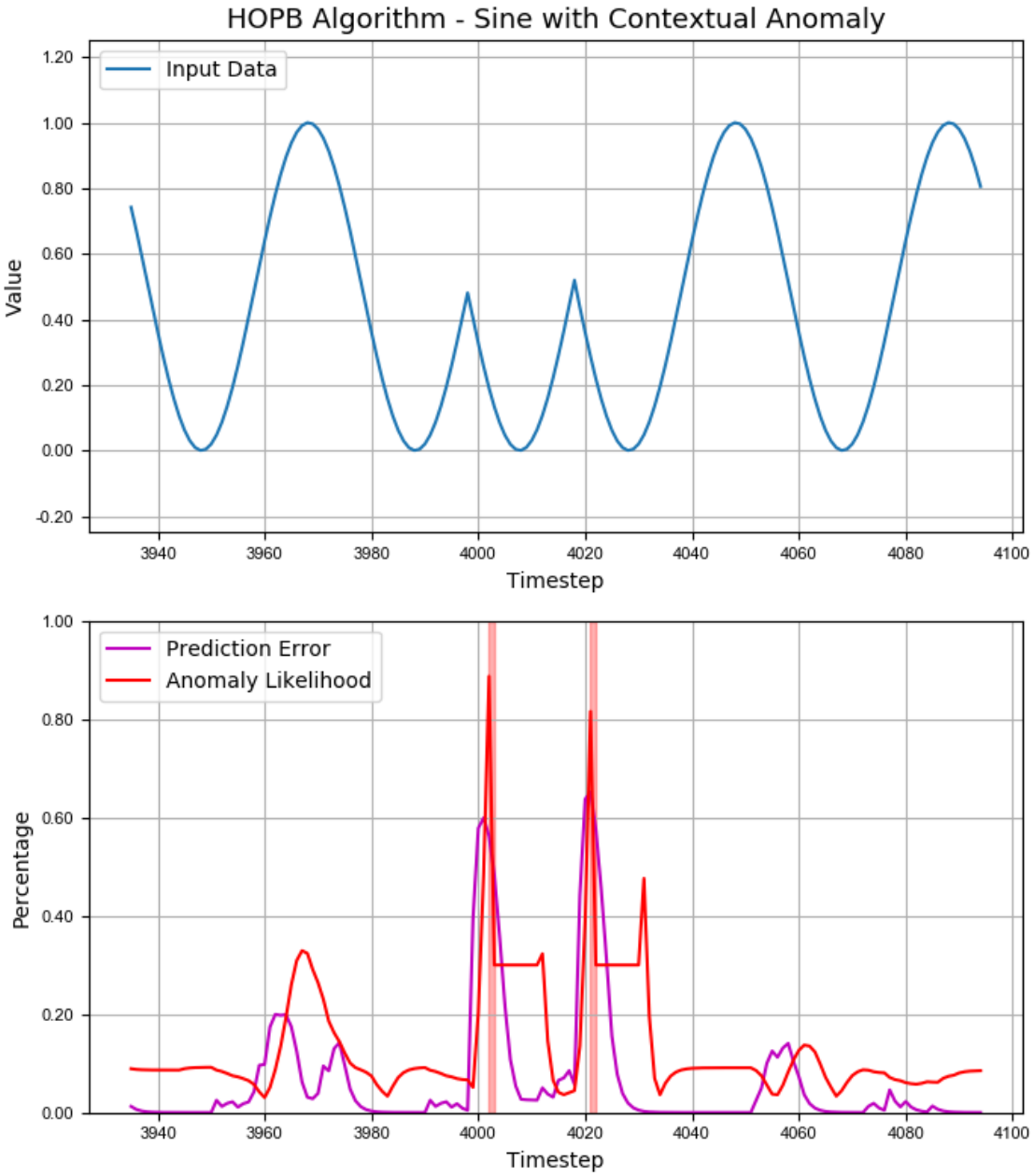

Figure 6.20: *Visualization of the HOPB algorithm's behavior in the presence of a contextual anomaly. Note the spreading of the error into the timesteps that are contextually anomalous both at the beginning and end of the sudden context shift. This extra response is able to bring the anomaly likelihood to a reasonable level.*

6.5.3 In the Presence of a Collective Anomaly

We present the HOPB algorithm with the same collective anomaly model as seen in Section 6.2.3. The results can be seen in Figure 6.21. As intended, the accumulative average mechanism captures the gradually increasing error and fills in the gaps of the erratic behavior seen in Figure 6.3. This along with the fact that the anomaly likelihood model has modeled lower prediction errors in the past has the effect of driving the anomaly probability up when the anomaly occurs.

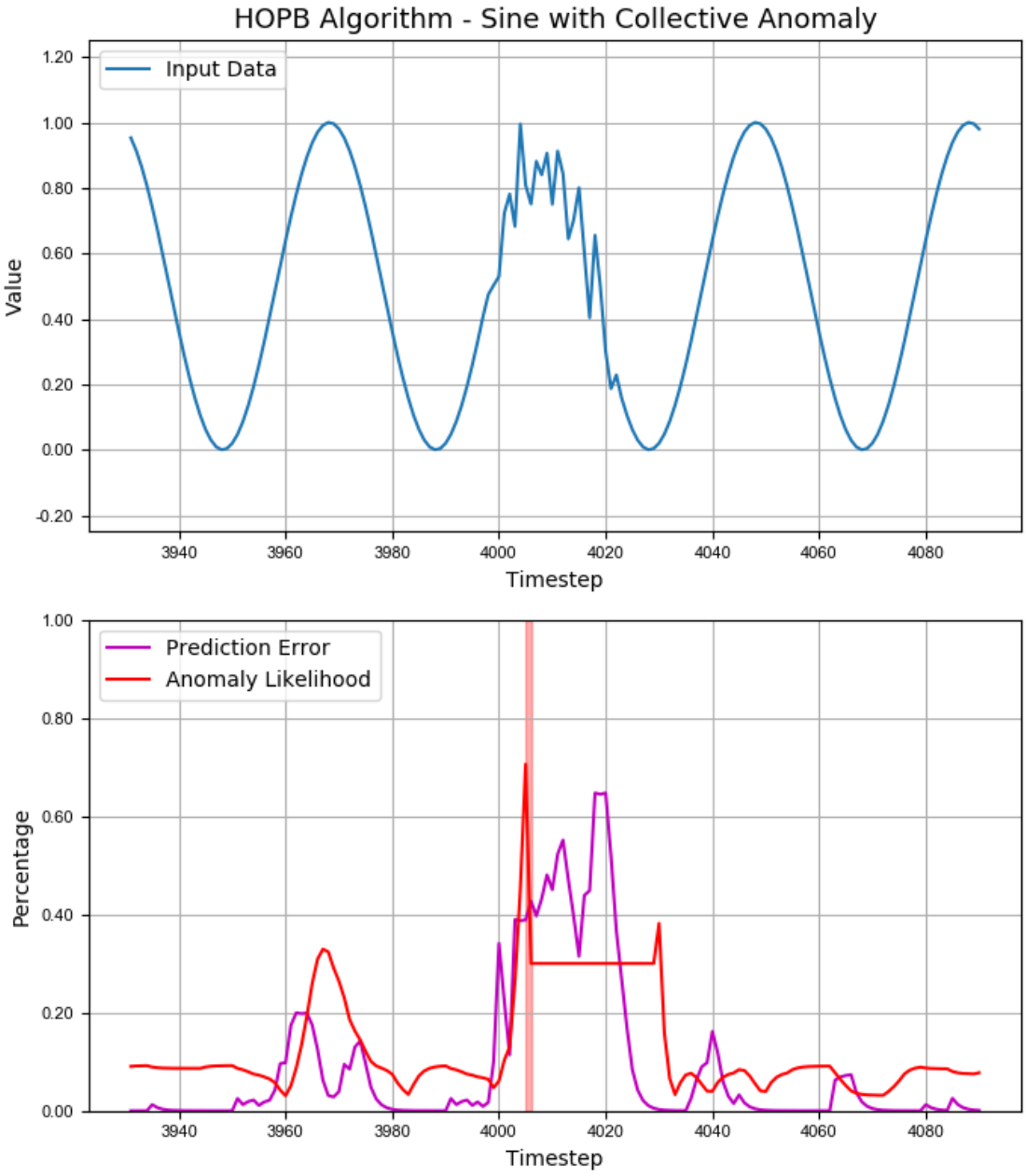

Figure 6.21: *Visualization of the HOPB algorithm's behavior in the presence of a collective anomaly. Note the accumulation of error during the collective anomaly is properly caught and isolated from the rest of the dataset which is able to bring the anomaly likelihood probability up to a significant level.*

6.5.4 Increased Fault Tolerance

We also demonstrate the potential for an increased sense of fault tolerance over the Numenta algorithm in this section. To demonstrate this, we chose a dataset with relatively large noise such that the underlying HTM model struggles with it. Of course, reducing bursts of error is useless if we also reduce the response during a true anomaly, so a spatial anomaly is injected into the dataset to see what happens. We show the results after the HTM network has sufficiently learned the sequence and prediction errors are stable. In Figure 6.22, on the upper left, we use only first-order predictions that return many random bursts of error due to faults in the network. On the upper right, we test the HOPB algorithm without the accumulating average so only the high-order predictions are having an effect. The bottom plot features the entire HOPB algorithm including the accumulating average. We use reasonable HOPB parameters for this experiment: $n_{\max} = 10$, $\psi = 500$, $\alpha = 0.7$ and $\epsilon = 0.1$. We see in Figure 6.22 what averaging over multiple predictions per timestep can do for us in terms of mitigating random errors. Every burst of error that does not correspond to an actual anomaly essentially vanishes.

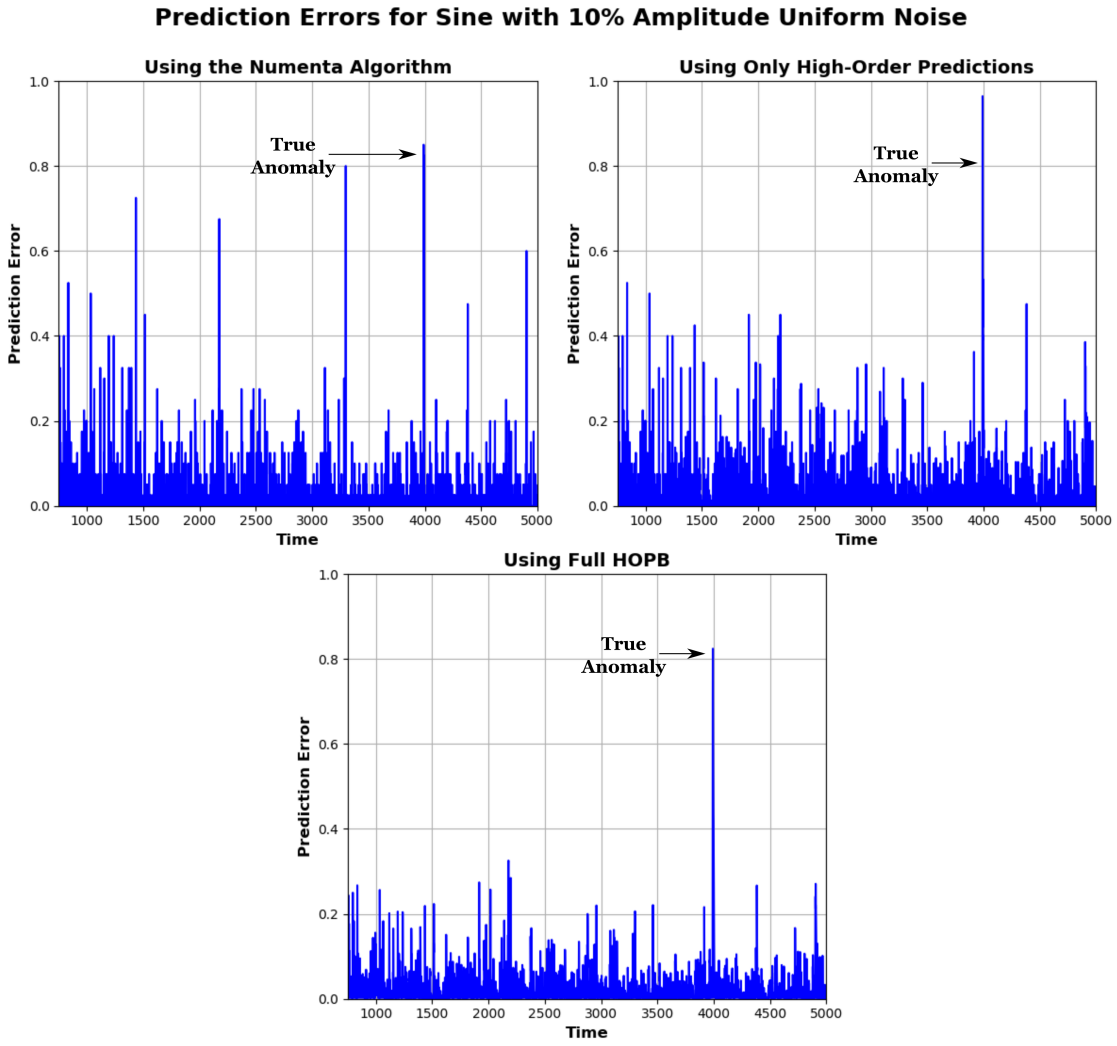

Figure 6.22: *Visualization of HOPB's potential for increased fault tolerance. This figure displays the prediction errors over time for a sine wave with uniform noise with 10% of the total range in amplitude after sufficient learning has taken place. We see on the upper right that averaging over several predictions made at distinct points in the past for each timestep alone significantly reduces the impact of random faults in the network seen to occur when using only first-order predictions. This increases the visibility of the true anomaly which also creates high prediction error. The bottom plots shows us how using the accumulating average is able to better pick out and isolate only the strongest responses.*

When using HOPB, we query the model at several different points in time for each timestep. Any random fault would essentially need to occur the same way at several different points in the past to have a significant impact on the end result. This is highly unlikely, supported by the upper right plot in Figure 6.22. The only time where the

prediction error remains high is when *no* past predictions could have realistically guessed the value at any point. That is to say when a true anomaly occurs.

We also see in the bottom plot of Figure 6.22 that using the accumulating average has the additional benefit of adding another layer of fault tolerance to the algorithm. The accumulating average helps wash away those high prediction errors that aren't immediately followed by at least one or more adjacent high prediction errors. The accumulating average helps us to automatically focus only on the anomalous spots that have the most evidence to be an anomaly given by multiple adjacent large errors. We see in Figure 6.22 that the largest erroneous fault is less than half of the prediction error of the true anomaly. This is a large performance benefit over using only first-order predictions.

Testing Consistency Upon removing the anomaly, reducing the noise and increasing the number of timesteps by four, we still see a noticeable difference of performance in Figure 6.23. In fact, using the Numenta algorithm generates several false positive events of relatively high error that are eliminated when using HOPB. In this figure, we are similarly using the entire HOPB framework with the same parameters used in Figure 6.22. We lastly visualize the number of predictions called during this experiment converge to approximately six to seven predictions in Figure 6.24.

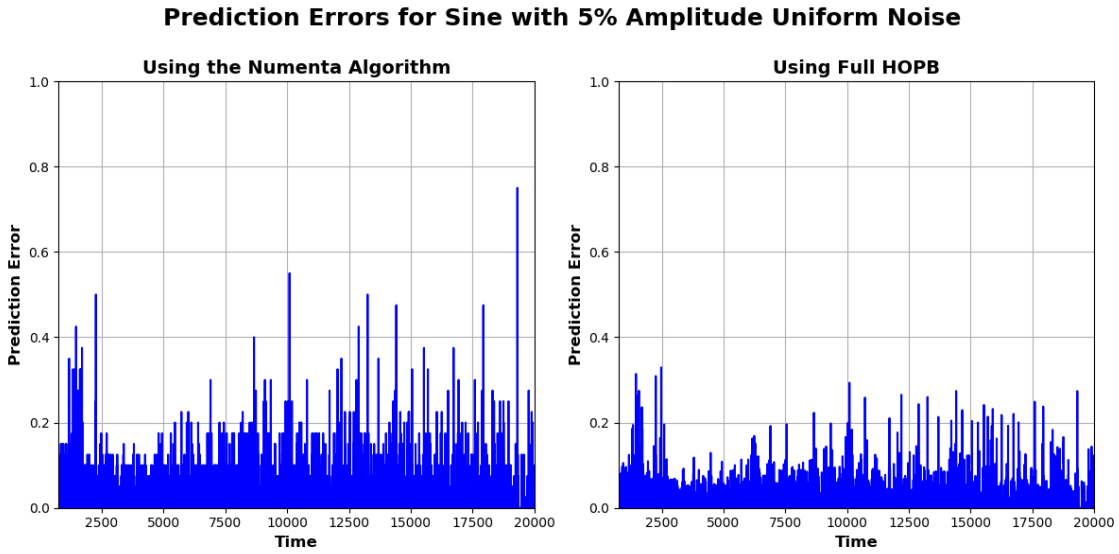

Figure 6.23: *Observation of the increased fault tolerance effect of the full HOPB framework appearing consistently across 20,000 timesteps of a sine wave with uniform noise with 5% of the total range in amplitude after sufficient learning has taken place. The Numenta algorithm on the left generates several false positive events of high error that are erased by HOPB.*

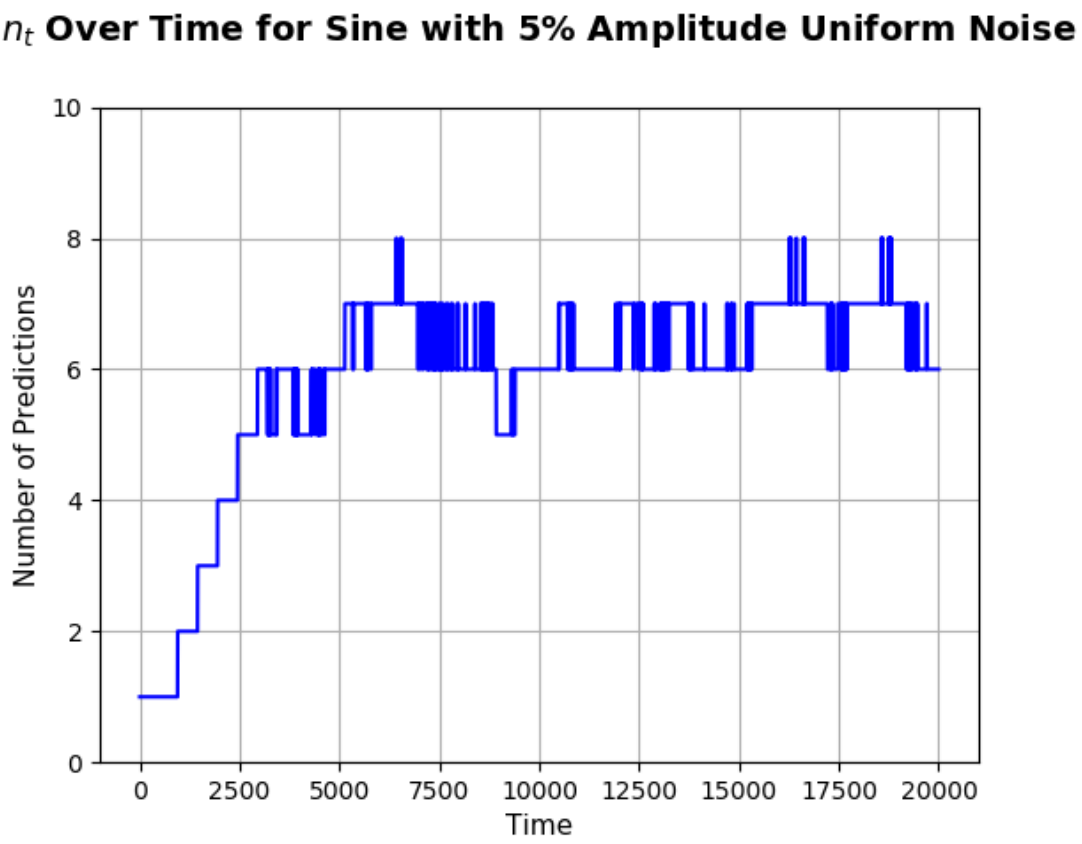

Figure 6.24: *The convergence of n_t to approximately six to seven predictions during the consistency experiment.*

6.6 Comparative Performance on External Datasets

In the effort to increase the credibility and reliability of the above results, we test HOPB and the Numenta algorithm on several external datasets from various domains where streaming data is common. We use these as illustrative examples of the benefit HOPB provides as well as a statement about the general purpose value it has to a wide variety of domains. The results are shown in this section.

Note also that HTM requires a small chunk of the data in the beginning to learn which is often referred to as the probationary period. The timestep labels in the results plots of this section reflect the results on non-probationary period data thus they often don't start at 1.

6.6.1 Electromyogram

The details for this dataset can be found in Section 5.2.3. We visualize the EMG readings in Figure 6.25. Note the cease of electrical activity from the muscle late into the dataset. That is our target anomaly.

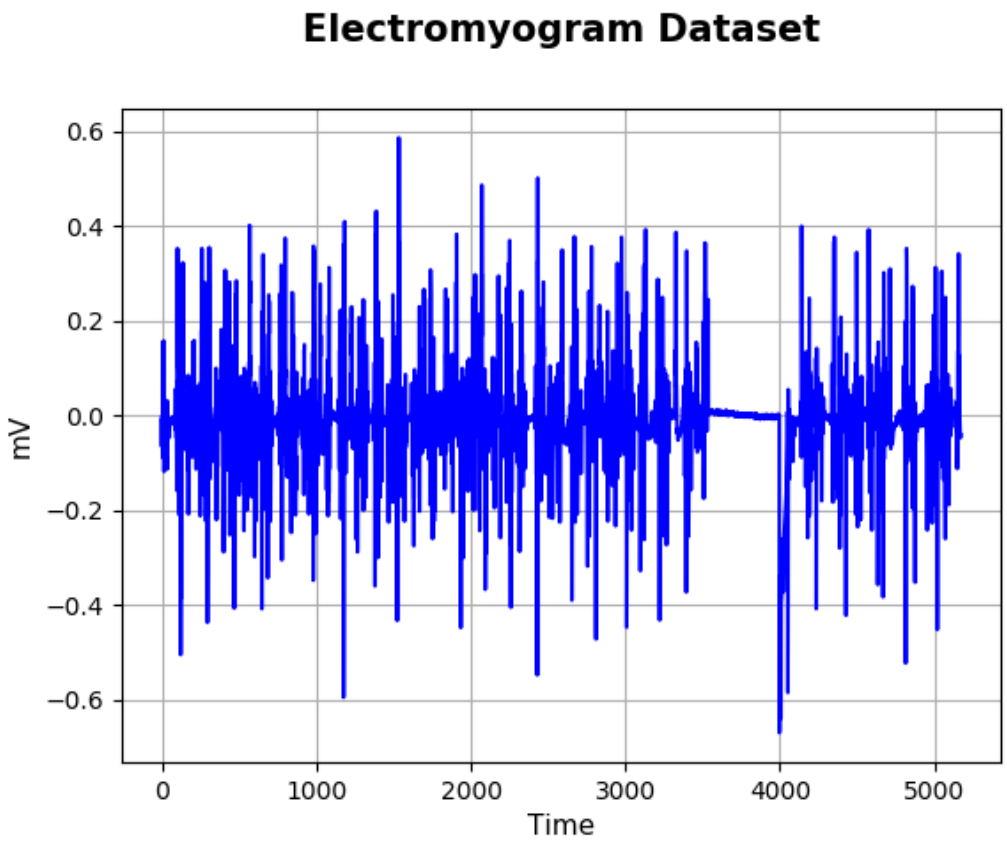

Figure 6.25: *Visualization of the electromyogram dataset. The dataset is highly unpredictable in general. The cease of electrical activity late into the dataset is our target anomaly which requires a sophisticated sense of a contextual local standard deviation of readings.*

Note that this anomaly consists of two main contextual shifts where the noisy signal turns into a relatively flat line and when the noise returns. Both the Numenta algorithm and using HOPB were able to detect the first contextual shift signified by the sudden *absence* of prediction error. The sudden absence of prediction error bring the anomaly likelihood up to significant values similar to the sudden appearance of it.

6.6.1.1 Under Normal Conditions

One can easily see a lightweight statistical method such as a moving average would fail for this dataset. When the muscle is functioning normally, the the precise values are highly noisy and unpredictable in general. A time modeling algorithm must be extremely tolerant of noise and be aware of a contextual local standard deviation of the readings to detect the anomalousness of the temporary halt of electrical activity. Recall these measurements are taken with a rapid sampling frequency with the whole dataset spanning only a few seconds. Encoding timestamps doesn't make sense for this dataset, so naturally any timestamp encoding is omitted.

We first visualize the reported prediction errors over time shown in Figure 6.26. The HOPB parameters used here are $n_{\max} = 10$, $\psi = 600$, $\alpha = 0.75$ and $\epsilon = 0.15$. The parameters are set in favor of easily allowing high orders of prediction because the high unpredictability of the dataset would naturally inhibit them. In particular, ϵ is needed to be set quite high simply to illustrate the effect of using high-orders of prediction. In this case, somewhat aggressively allowing high orders results in a fast benefit but this is not always true. It is generally safer to choose more strict parameters and in a true streaming environment the number of timesteps processed will be much higher thus waiting slightly longer for reliable orders of prediction would not be an issue.

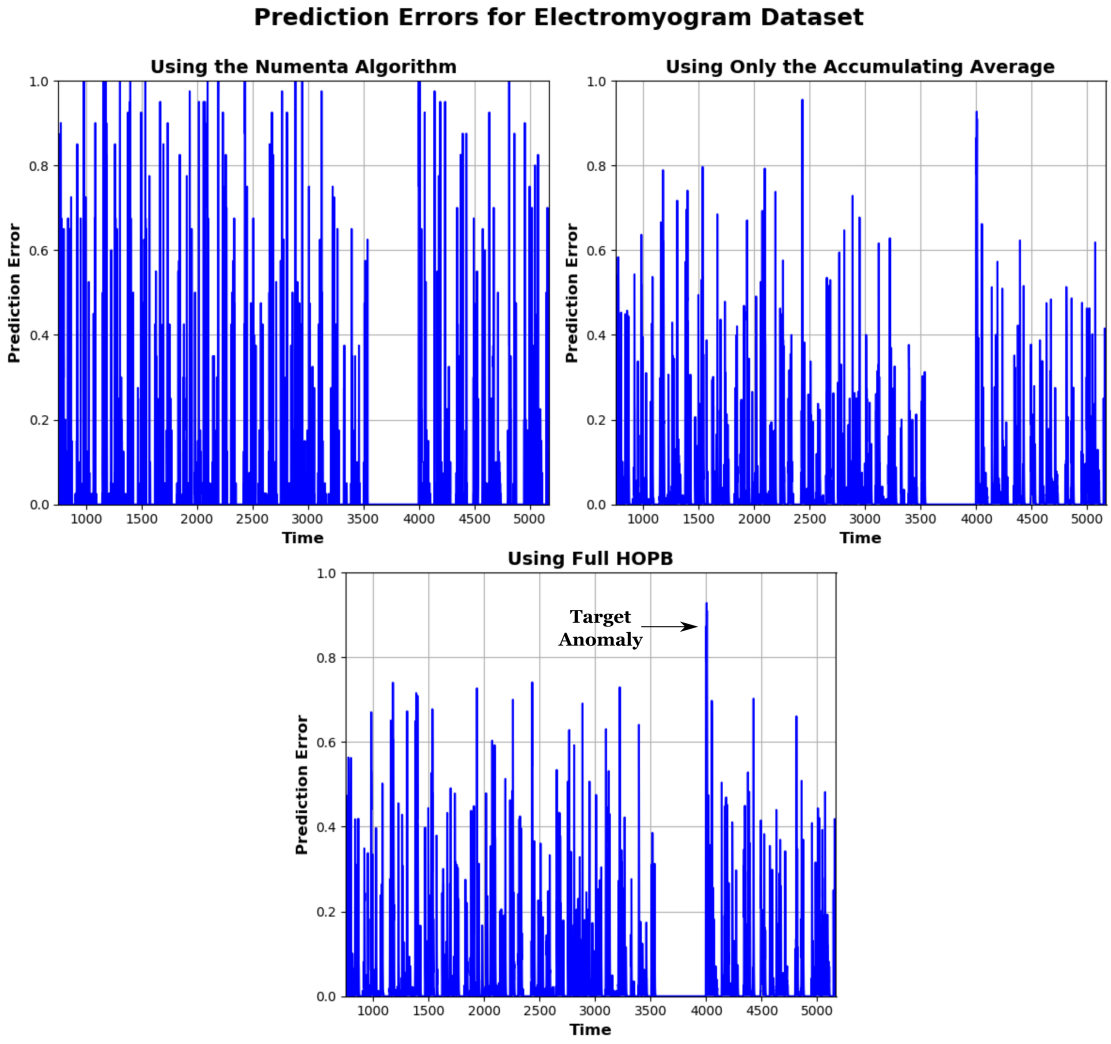

Figure 6.26: *Prediction errors over time on the electromyogram dataset. The upper left plot is the Numenta algorithm which only uses first-order predictions. The upper right plot applies the accumulating average over without any high-order predictions which we see is not enough to isolate the second context shift. The bottom plot uses the full HOPB algorithm which incorporates high-order predictions governed by a dynamic chain size.*

See in Figure 6.26 the performance benefit when using HOPB with respect to the ability to discern true anomalies from random faults. When using the Numenta algorithm, it is virtually impossible to discern between random faults and the second context shift. As we include high-order predictions and the full HOPB algorithm including the accumulating average, we can start to see the difference between faults and the target anomaly. We see in the upper right plot that using the accumulating average alone is not enough to separate normal behavior versus the target anomaly. In the bottom plot, which adds

high-order predictions, not only do we see that separation but we see more consistent behavior outside of the anomalies which is desirable for modeling purposes. During the first context shift, we see an absence of prediction error when the data becomes a flat line and accordingly becomes very predictable. This would drive the anomaly likelihood up in all three cases. When the electrical activity resumes, HOPB is able to distribute the prediction error and uniquely isolate that anomalous behavior as predictions of the high-order future predict a continued flat line. We can ensure high-order predictions are being used in Figure 6.27 which shows the number of predictions rising and converging to approximately three predictions.

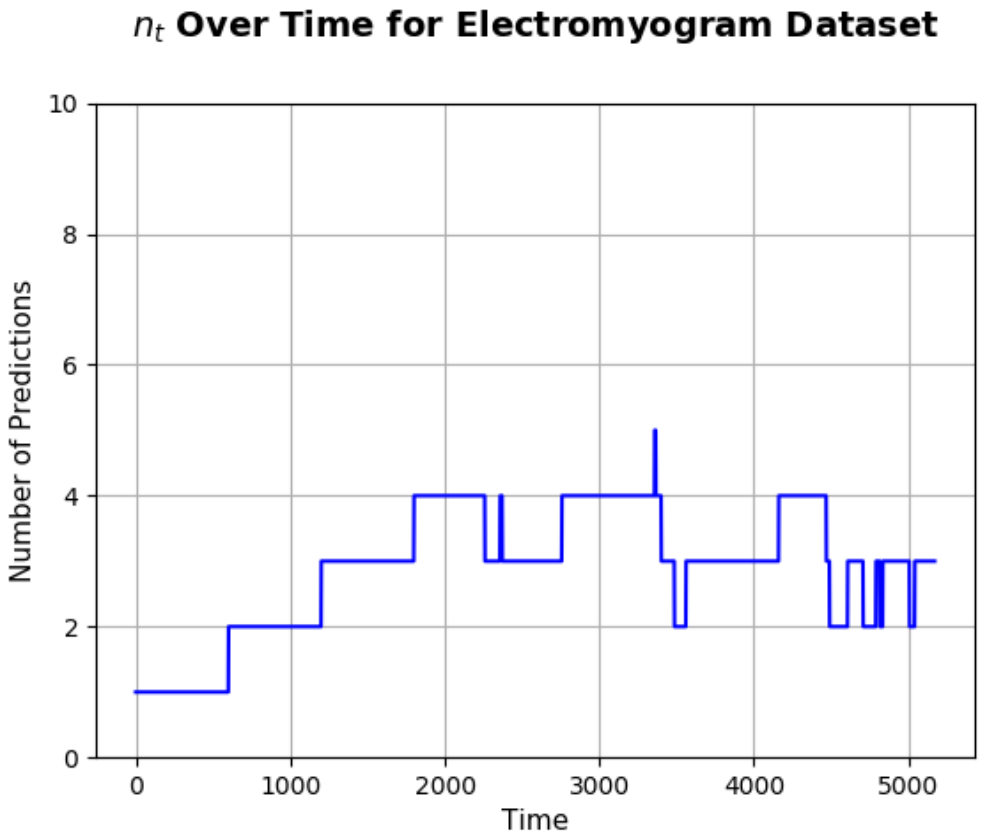

Figure 6.27: *Visualization of the number of predictions called for floating up and down when using HOPB on the electromyogram dataset. The algorithm stays stable at around two and three predictions per timestep.*

6.6.1.2 Tweaking the Encoder

Figures 6.26-6.27 show the results when initializing the underlying HTM model the normal way by choosing the globally expected minimum and maximum value the data can take on with a small amount of padding. Ideally, with some human foresight, we can recognize the data signal is going to be very unpredictable and we can manually initialize our model to be less sensitive to the exact value of the signal within its bounds. We can "cheat" this way by manually establishing much larger minimum and maximum bounds for the underlying HTM encoder. You can think of this like "zooming out" and thus internally representing less about the exact value of the signal. This should only be attempted if the engineer happens to know the internal values of the signal within its bounds are going to be naturally unpredictable and not follow any discernible pattern such as in this dataset. Figure 6.28 displays the result of this action.

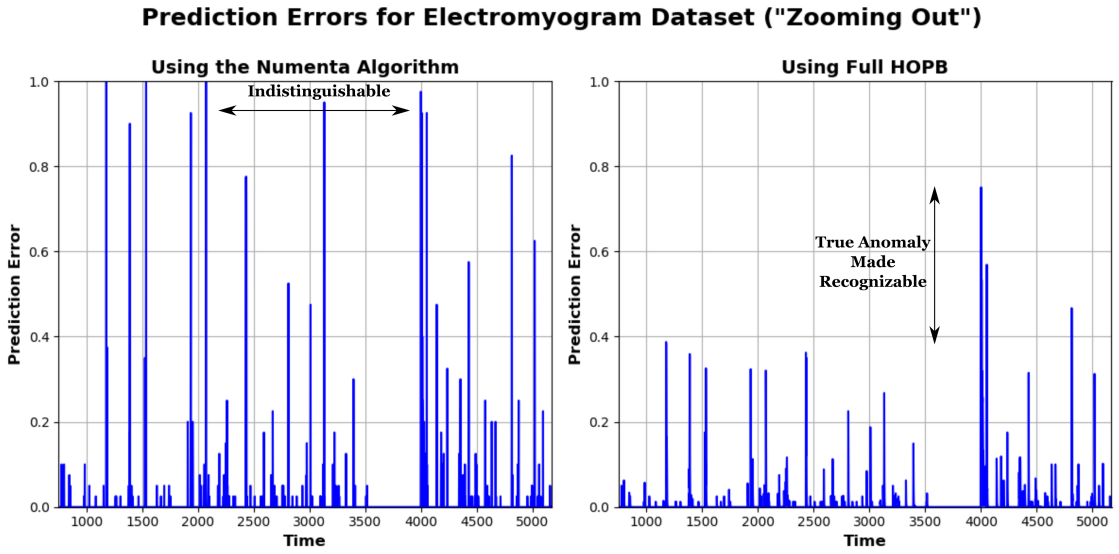

Figure 6.28: *Prediction errors over time on the electromyogram dataset after manually adjusting the maximum and minimum bounds in the HTM encoder. The left plot shows the Numenta algorithm where determining fault from the target anomaly is impossible. The right plot shows HOPB which correctly separates faults from the target anomaly.*

Upon doing this, we can see a much smaller average prediction error in Figure 6.28 but it does not protect us against the random faults of the network when using the Numenta algorithm. When using HOPB, we set the parameters to quickly reach a high number of predictions to show their benefit. The exact parameters are $n_{\max} = 10$, $\psi = 200$, $\alpha = 0.6$ and $\epsilon = 0.15$. We see on the right the benefit of using HOPB is made even greater with

the target anomaly bearing little resemblance to faults. Note in Figure 6.29 that high-orders of prediction specifically give us the insight we need to determine this. Each order experiences its own faults but once averaged together, only the prediction error during the target anomaly remains large. Lastly, Figure 6.30 shows the number of predictions quickly reaching and converging at ten predictions as desired.

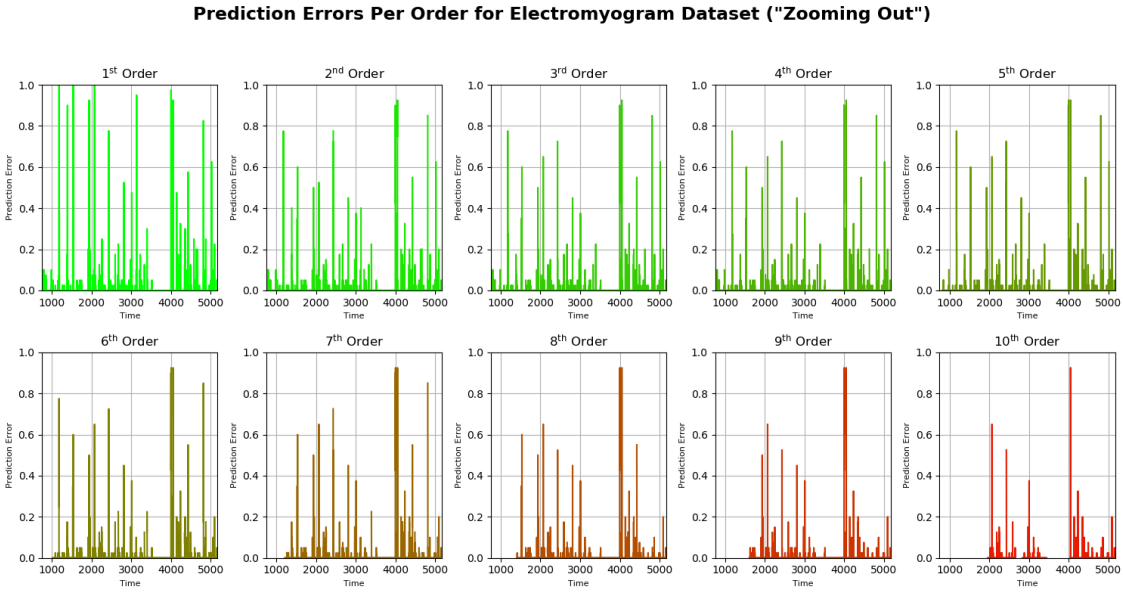

Figure 6.29: *Prediction errors over time on the electromyogram dataset per order while using HOPB after adjusting minimum and maximum bounds in the HTM encoder. Each order of prediction experiences faults but once averaging all orders together per timestep only the target anomaly remains.*

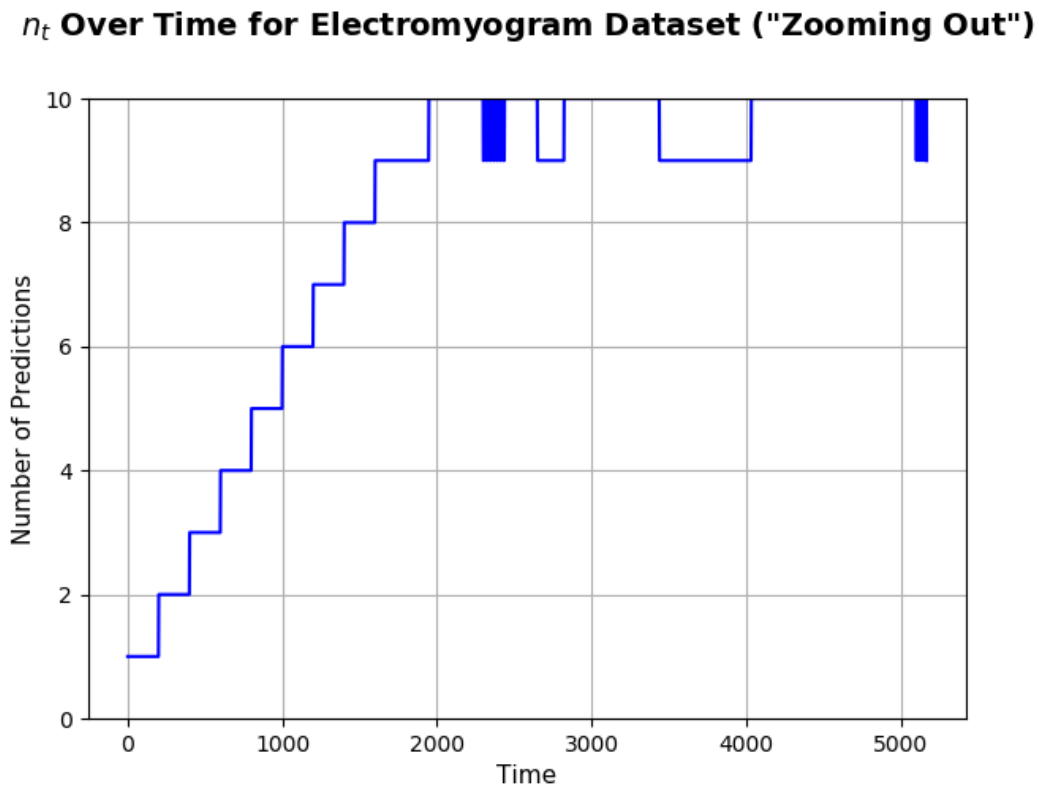

Figure 6.30: *Visualization of the number of predictions called for floating up and down when using HOPB on the electromyogram dataset after adjusting minimum and maximum bounds in the HTM encoder. HOPB converges to ten predictions.*

6.6.2 Yahoo! Production Traffic

As described in Section 5.2.3, included in this section are the results from an example of the data included in the Yahoo! Benchmark for anomaly detection [61]. The production traffic dataset is visualized in Figure 6.31. Notice the very short-term spatial anomaly. The maximal peak of the spatial anomaly consists of only one timestep. The timestamps provided on this dataset are simple increasing integers and don't artificially provide any cyclic insight. Thus, they were discarded for our experiment.

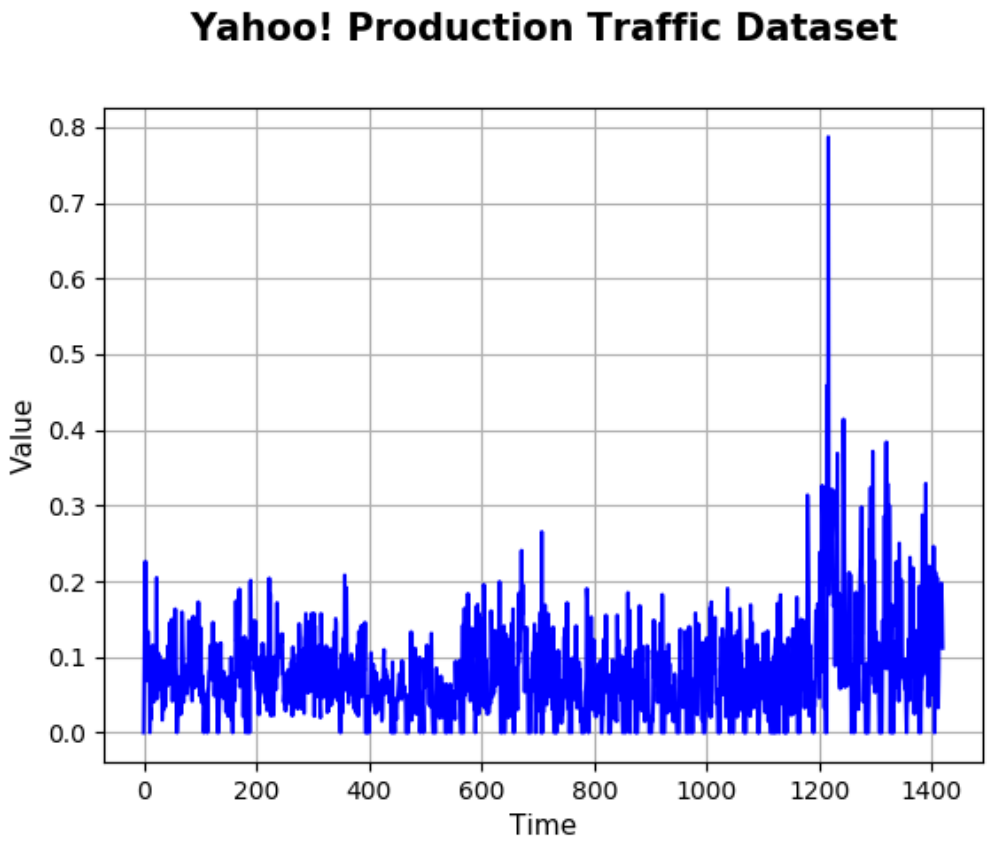

Figure 6.31: *Visualization of the Yahoo! production traffic dataset. This data consists of real production traffic recorded on Yahoo! properties. The traffic is noisy with a clear spatial anomaly late in the dataset.*

We set the HOPB parameters very much in favor of quickly allowing high orders of prediction because the dataset is fairly small. We wish to illustrate the value of HOPB as best as possible. The exact parameters used are $n_{\max} = 10$, $\psi = 100$, $\alpha = 0.6$ and $\epsilon = 0.05$. The resulting reported prediction error over time is shown in Figure 6.32.

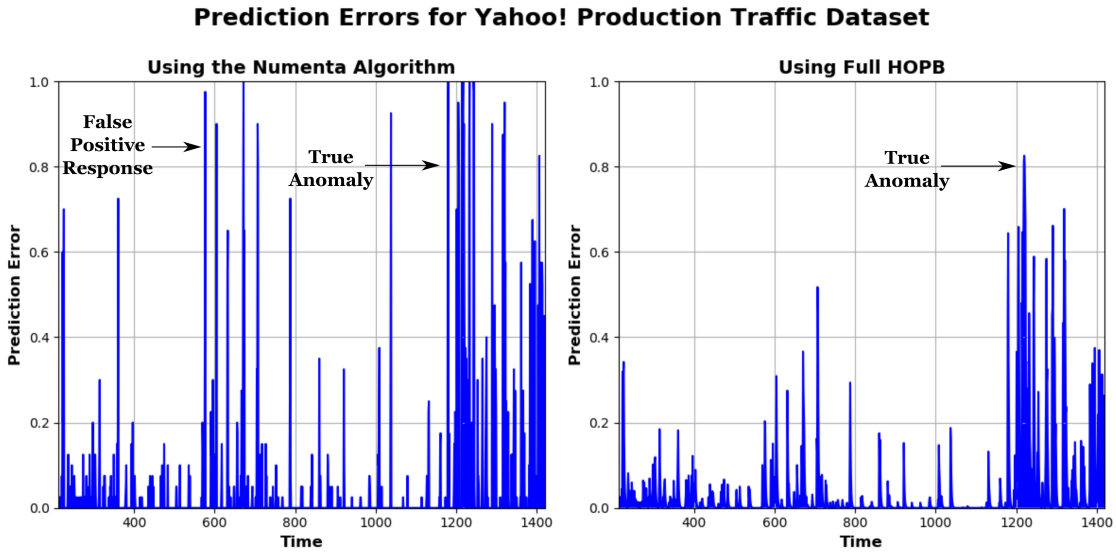

Figure 6.32: *Prediction Errors over time for the Yahoo! production traffic dataset. Notice the increased size of the relative gap in prediction error between the true anomaly and the closest fault when using HOPB.*

Similar to the other experiments in this thesis, HOPB is able to better discern between random fault and true anomaly as seen in Figure 6.32. We see high orders of prediction providing the additional information we need in Figure 6.33. Lastly, Figure 6.34 shows the number of predictions climbing to the maximum amount before falling due to the presence of increased uncertainty.

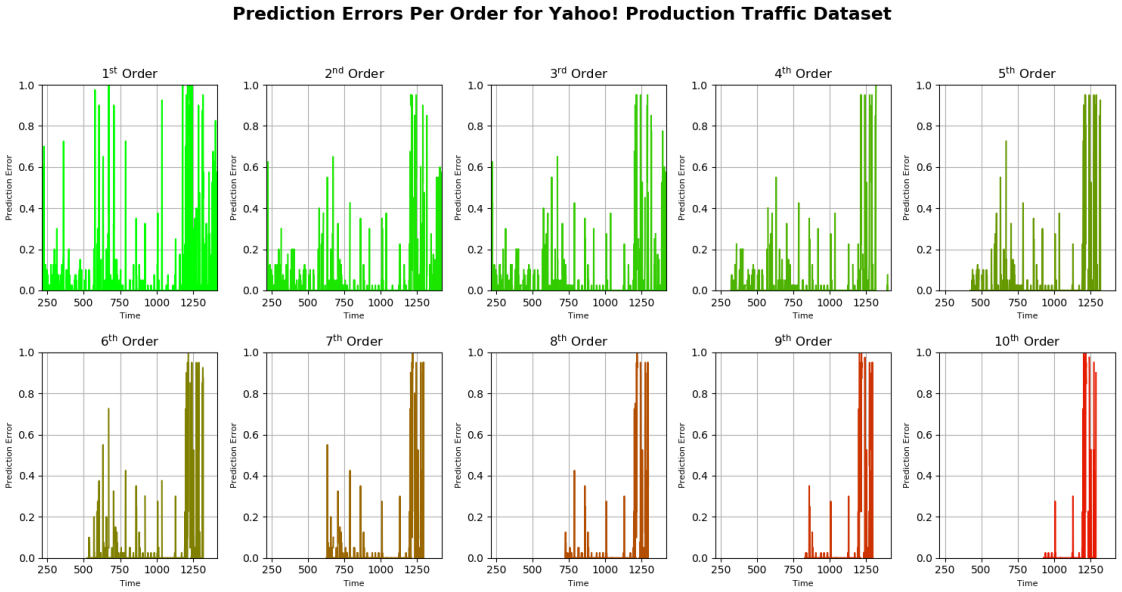

Figure 6.33: *Prediction Errors over time per order for the Yahoo! production traffic dataset. Notice the high-orders of prediction providing the needed information for better fault and anomaly discernment.*

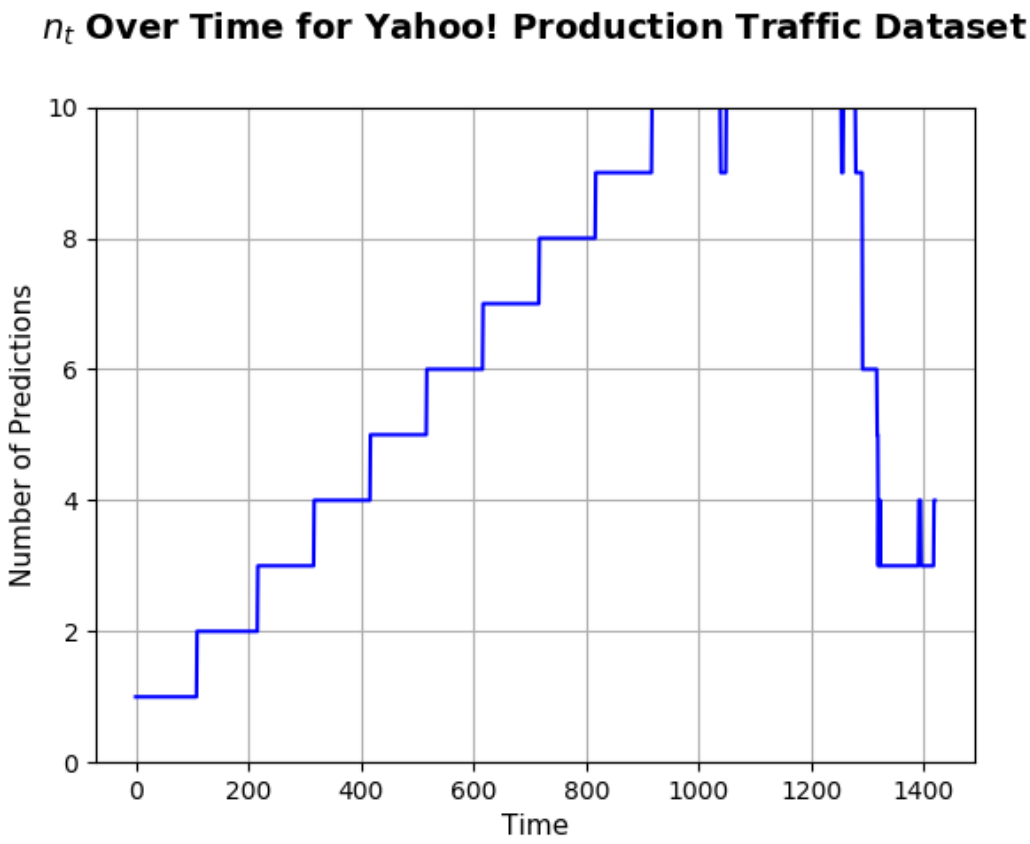

Figure 6.34: *Visualization of the number of predictions called for floating up and down when using HOPB on the Yahoo! production traffic dataset. The quick climbing action occurs as the HTM model learns the underlying pattern followed by a quick drop as the uncertainty increases.*

6.6.3 Arterial Pressure Readings

As described in Section 5.2.3, included in this section are the results from the arterial pressure dataset provided by the Department of Physiology at Michigan State University [67]. We can visualize a subset of this dataset if Figure 6.35. Specifically, we visualize what happens when a premature ventricular contraction occurs and the blood pressure reading is thrown off it's normal course. A reading is taken every 0.2 seconds for this data. The timestamps offer no cyclic information and thus they are not used in any way.

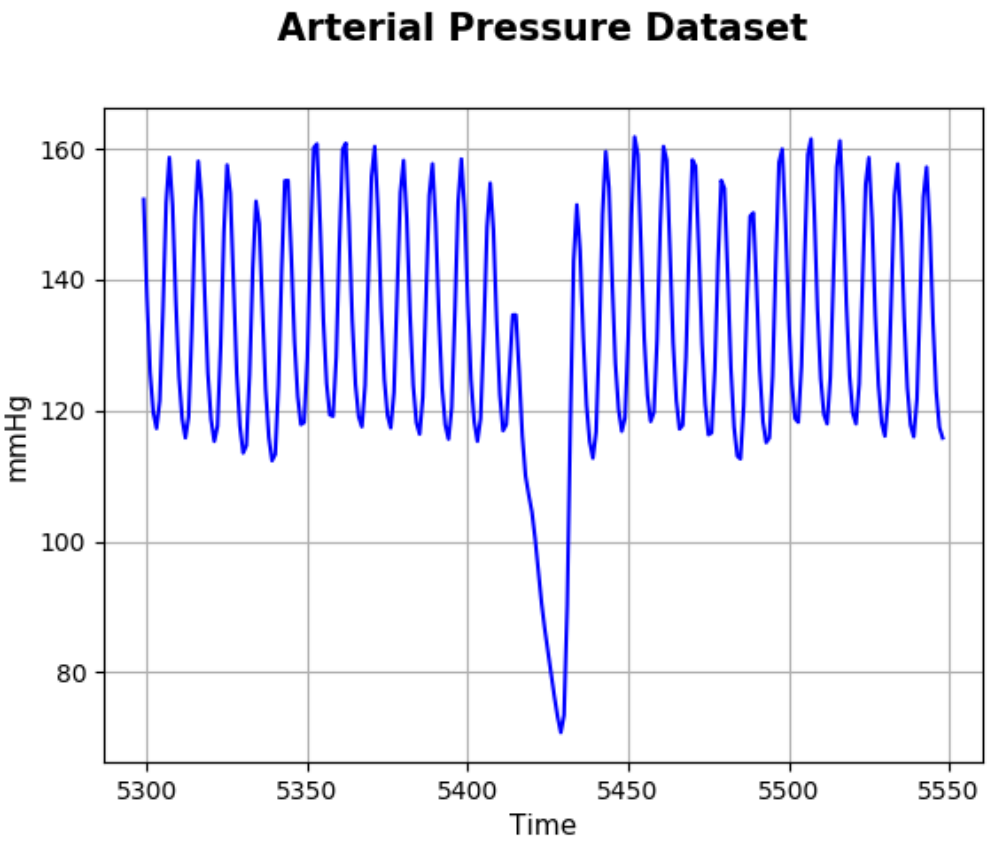

Figure 6.35: *Visualization of the arterial pressure dataset during a premature ventricular contraction. Note that this anomaly contains spatial deviation but also contextual deviation as the anomaly is spread through many timesteps which increasingly diverge from normal signal behavior.*

In this dataset, the premature ventricular contraction brings the minimum value of the arterial pressure to a much lower value than what is typically seen in the rat under normal conditions. The plunge in mmHg is distributed across many timesteps that increasingly diverge. If we establish the minimum value when building the HTM encoder as this minimum value seen during the anomaly, both the Numenta algorithm and HOPB can easily detect the anomaly because a large portion of the encoder would never be used for the majority of the dataset. However, to do this is to cheat in a sense because it assumes we know the anomaly is going to occur. Instead, we establish the minimum and maximum bounds in the HTM encoder to be what is seen during non-anomalous conditions. Additionally, we test two different random initializations of the underlying HTM network to witness two separate benefits of using HOPB. We use HOPB parameters that aggressively allow high-orders of prediction to show their benefit in both initializations. The exact parameters for both experiments are $n_{\max} = 10$, $\psi = 500$, $\alpha = 0.8$ and $\epsilon = 0.15$.

6.6.3.1 Random Initialization #1

In the first random initialization we simply use the random seeds available by default in the NuPIC repository. We first visualize the prediction errors over time in Figure 6.36. We see in Figure 6.37 that HOPB converges at approximately three orders of prediction for these parameters on this dataset.

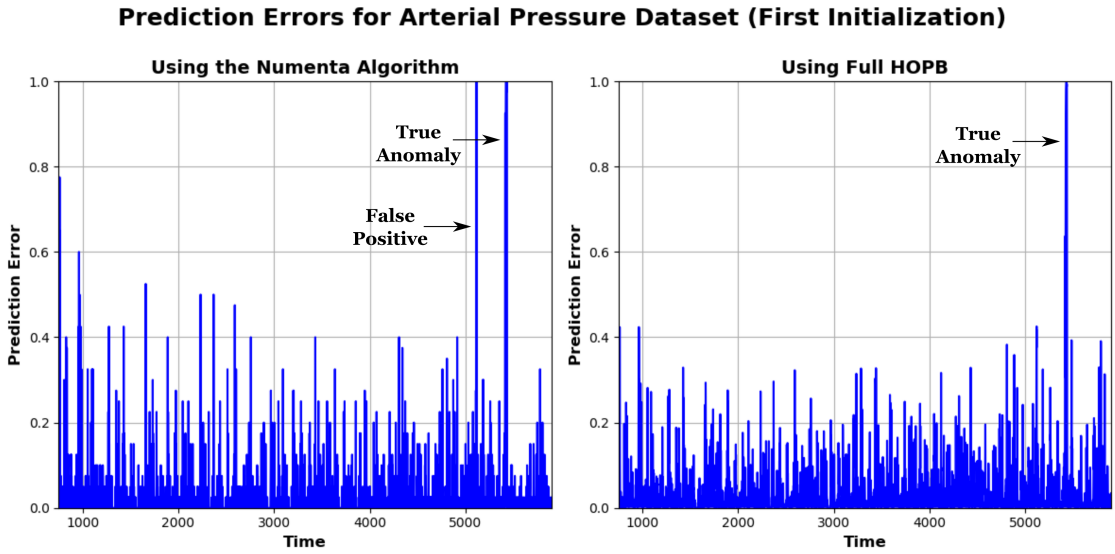

Figure 6.36: *Prediction errors over time on the arterial dataset with the first random initialization. Notice the presence of a false positive near the true anomaly when using the Numenta algorithm.*

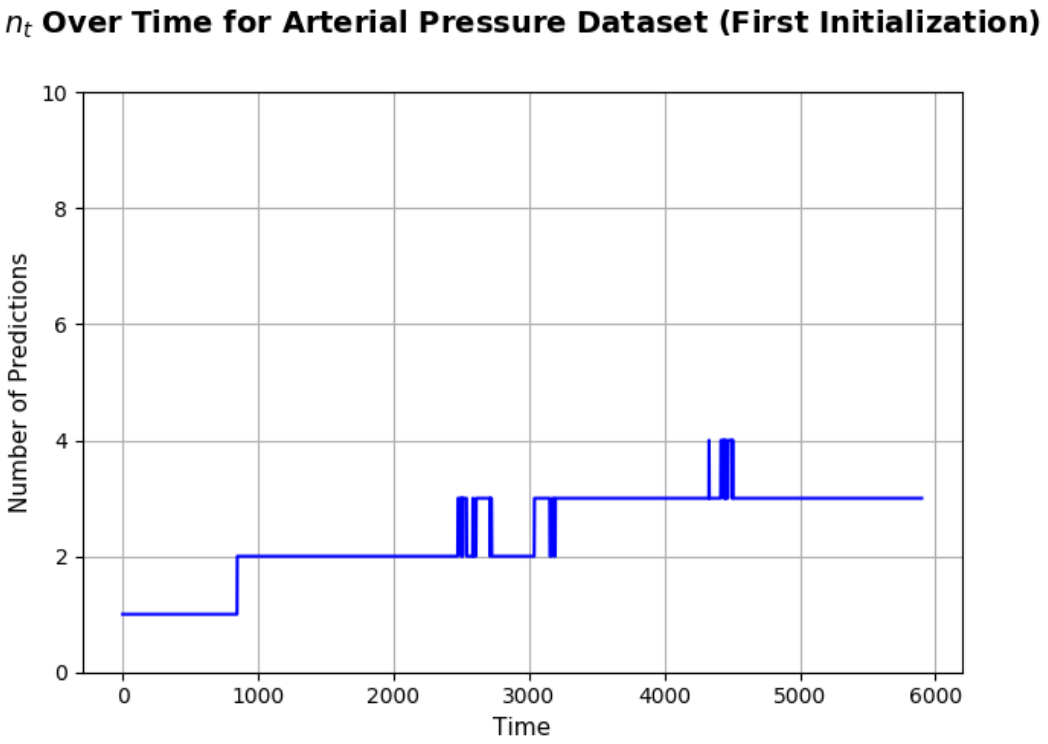

Figure 6.37: *Visualization of the number of predictions floating up and converging to three orders during the processing of the arterial pressure dataset with the first random initialization.*

We see in Figure 6.36 that using the Numenta algorithm results in a false positive near the true anomaly that is essentially completely avoided when using HOPB. Importantly, let us visualize how these differences in prediction error influence the anomaly likelihood modeling procedure. In this case, we are concerned what happens with the anomaly likelihood when the false positive response occurs. In Figure 6.38, we visualize what happens when using both algorithms side by side.

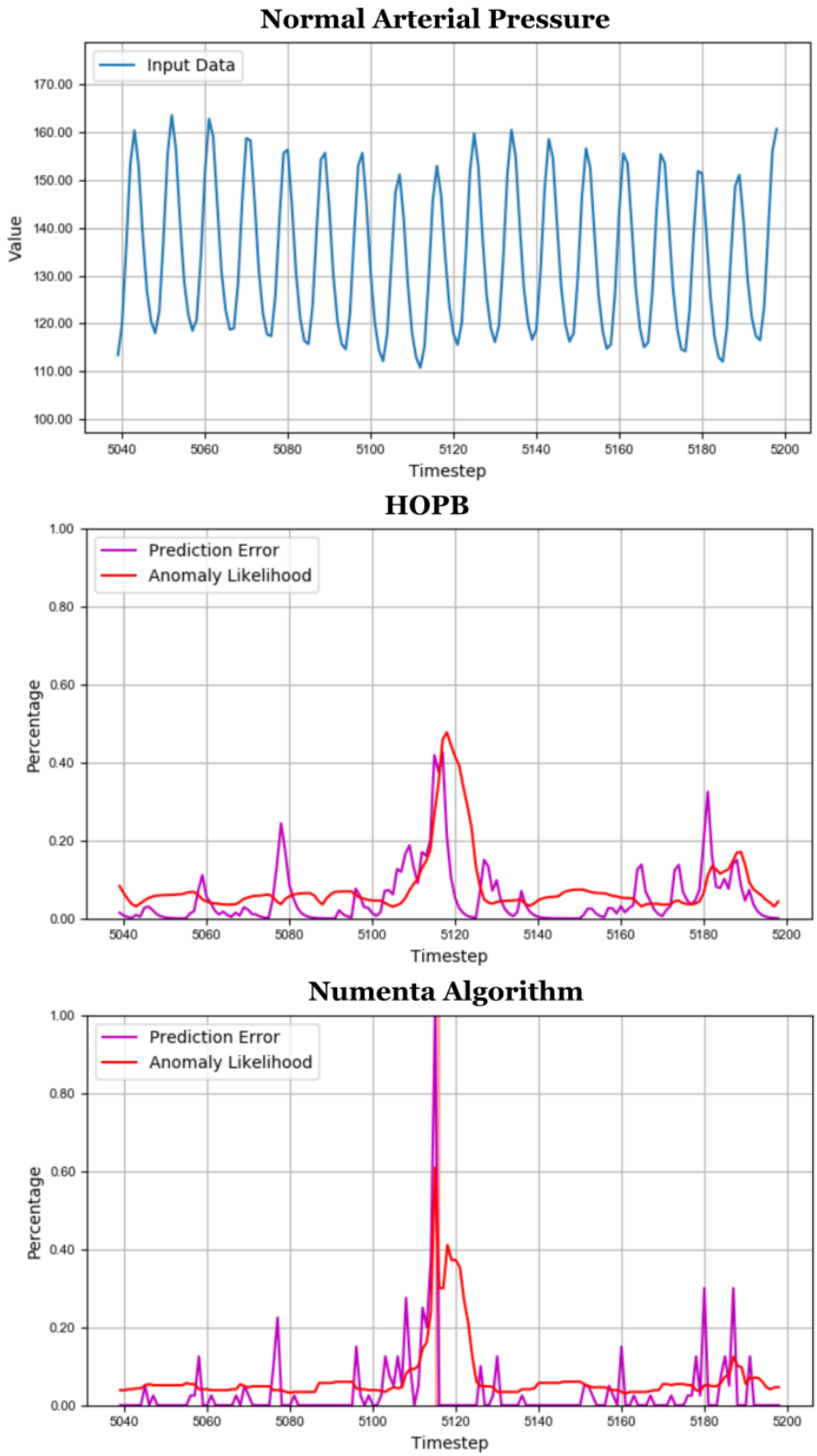

Figure 6.38: *Visualization of the anomaly likelihood when using the Numenta algorithm and HOPB when the Numenta algorithm has a false positive event with the arterial pressure dataset. Note that no recognizable anomaly occurs in the data yet the anomaly likelihood is pushed above 60% when using the Numenta algorithm. No recognizable anomaly occurs in the data and the anomaly likelihood stays below 50% when using HOPB.*

In Figure 6.38 we see that the erroneous spike in error does indeed cause a false positive response in the anomaly likelihood measure when using the Numenta algorithm. A typical threshold for an anomaly with the Numenta algorithm is around 55%. During the same subset of the data, we see what happens when using HOPB in the lower plot. Specifically, we see the anomaly likelihood stay below 50% which would not trigger an anomaly for any reasonable threshold.

6.6.3.2 Random Initialization #2

We arbitrarily switch up the random seeds to 9284 for the spatial pooler and 7538 for temporal memory for this experiment. It turns out that doing this highlights another key benefit of using HOPB. Firstly, we visualize the prediction errors over time in Figure 6.39. We see in Figure 6.40 that HOPB converges to approximately three to four orders of prediction for these parameters on this dataset with the same parameters.

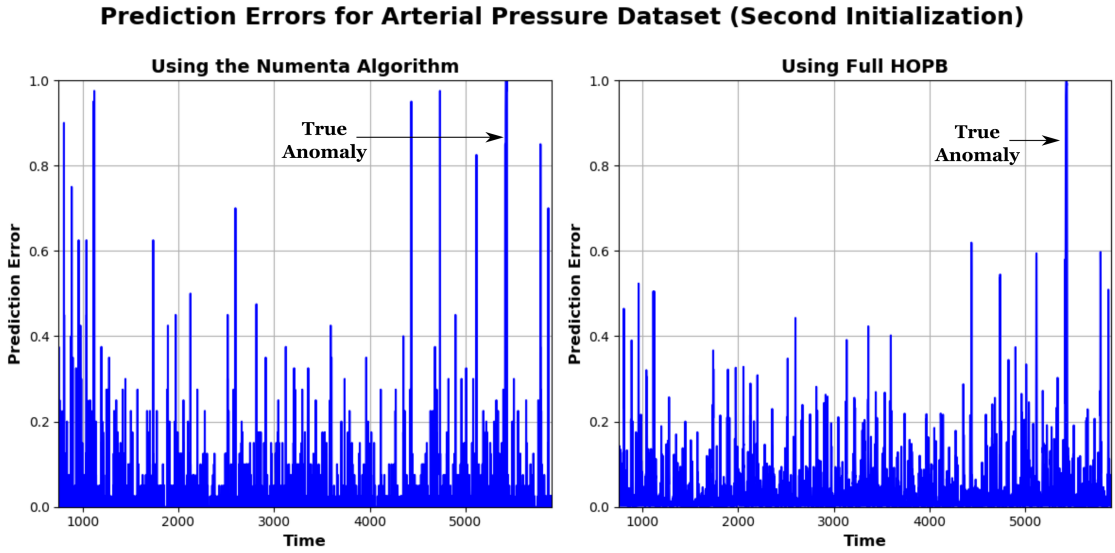

Figure 6.39: *Prediction errors over time on the arterial dataset with the second random initialization. Notice relatively stable performance provided by the HOPB algorithm. The relative difference between fault and true anomaly is made much more obvious when using HOPB*

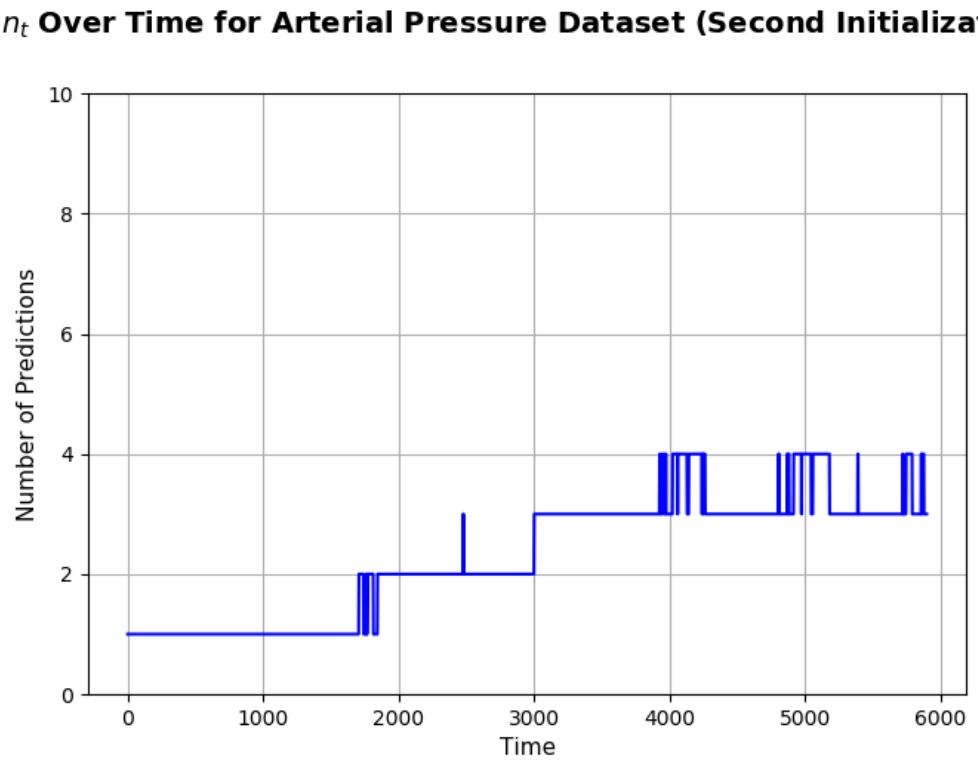

Figure 6.40: *Visualization of the number of predictions floating up and converging to three to four orders during the processing of the arterial pressure dataset with the second random initialization.*

The story we see in Figure 6.39 is similar to other experiments in this thesis. We get much more consistent performance from HOPB. This consistency has an important effect on the anomaly likelihood modeling procedure seen in Figure 6.41. We are specifically concerned with what happens during the true anomaly.

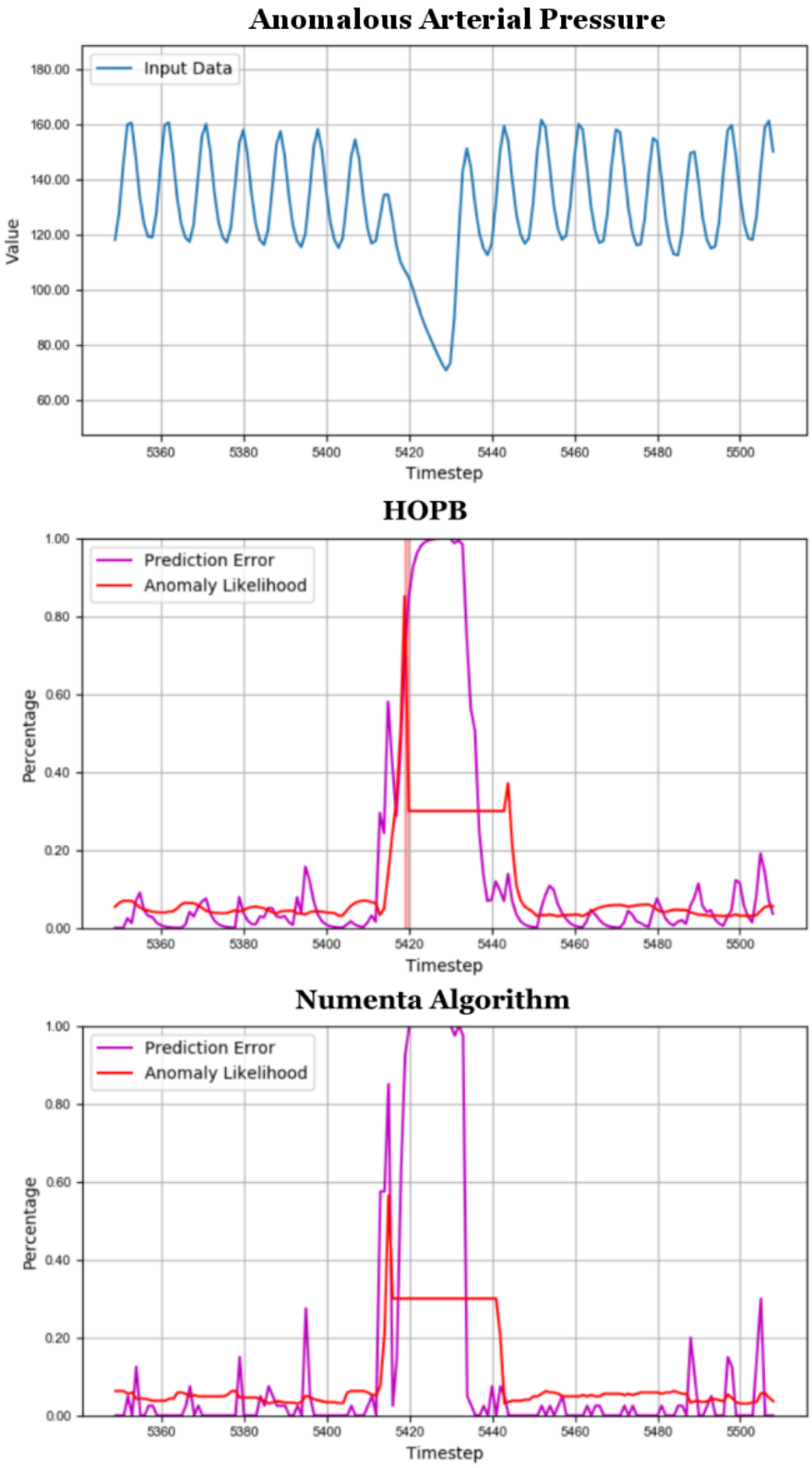

Figure 6.41: *Visualization of the anomaly likelihood when using the Numenta algorithm and HOPB during the true anomaly in the arterial pressure dataset. Note that despite the obvious anomaly in the data, the anomaly likelihood only reaches approximately 55% when using the Numenta algorithm. When using HOPB, the anomaly likelihood is brought to a significant value above 80%.*

The middle plot in Figure 6.41 shows us how despite the fact that the prediction error is brought to a maximal value, the anomaly likelihood fails to reach a significant value when using the Numenta algorithm. This is precisely because the rolling normal distribution that has formed earlier during processing has modeled several instances of high prediction error as a normal response. This brings down the reported likelihood that the target burst of high error is actually a true anomaly and not a random fault. The response with HOPB is much stronger in the bottom plot. The anomaly likelihood is brought to above 80% shortly after the true anomaly begins. This is because high error is much less common when using HOPB due to the averaging of multiple orders of prediction combined with the accumulating average. The rolling normal distribution that is formed while using HOPB is able to better discern the anomalousness of the true anomaly.

6.6.4 Annual Common Stock Prices

As described in Section 5.2.3, included in this section are the results from running HOPB on the annual average common stock prices in the United State from 1871 to 1970. We first visualize the data in Figure 6.42.

Figure 6.42: *Visualization of the annual common stock prices dataset. Notice the two anomalies in this data that occur during the late 1920s and after the 1950s.*

The timestamps for this data are clearly years but that does not conform to the timestamp encoder included in the HTM design in NAB so we discard the timestamps. Additionally, note that this dataset is exceedingly small. It only consists of 98 timesteps. In order to show the benefit of HOPB we must use parameters that very aggressively allow for high-orders of prediction. The exact HOPB parameters are $n_{\max} = 10$, $\psi = 1$, $\alpha = 0.0$ and $\epsilon = 0.02$. Despite essentially allowing all orders of prediction at every timestep, we find the change in prediction errors over time in Figure 6.43.

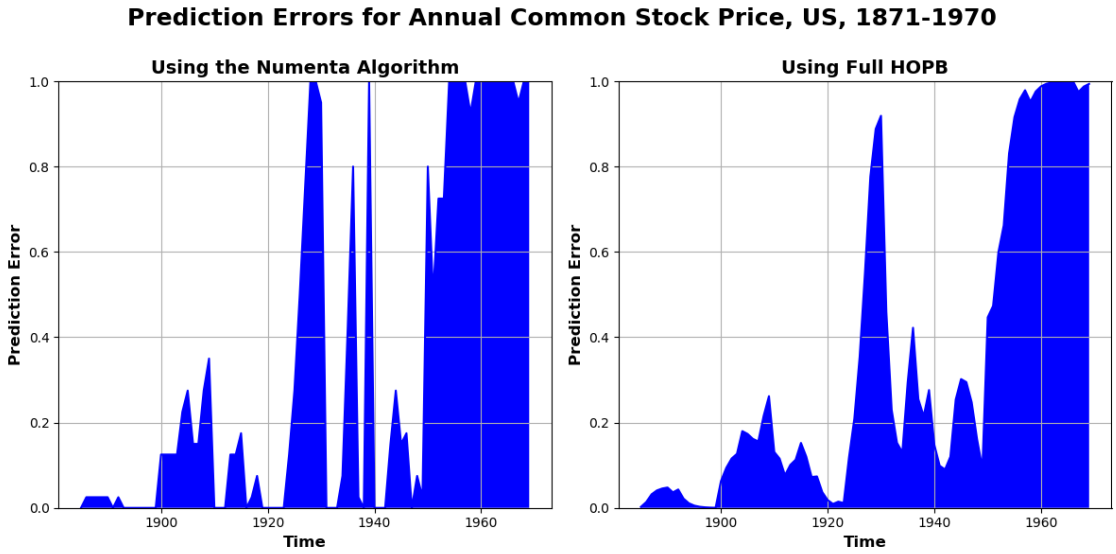

Figure 6.43: *Prediction errors over time on the annual common stock price dataset. Notice the two anomalies are much more easily discernible when using HOPB over only first-order predictions.*

We see in Figure 6.43 that using HOPB better captures the two true anomalies in the data. The prediction error during the late 1920s is consolidated into one event and the false positives that occur in the 1930s are virtually eliminated. See in Figure 6.44 the statistics per order when using HOPB. We are able to generate reliable and useful high-order predictions even with a dataset that is under 100 timesteps.

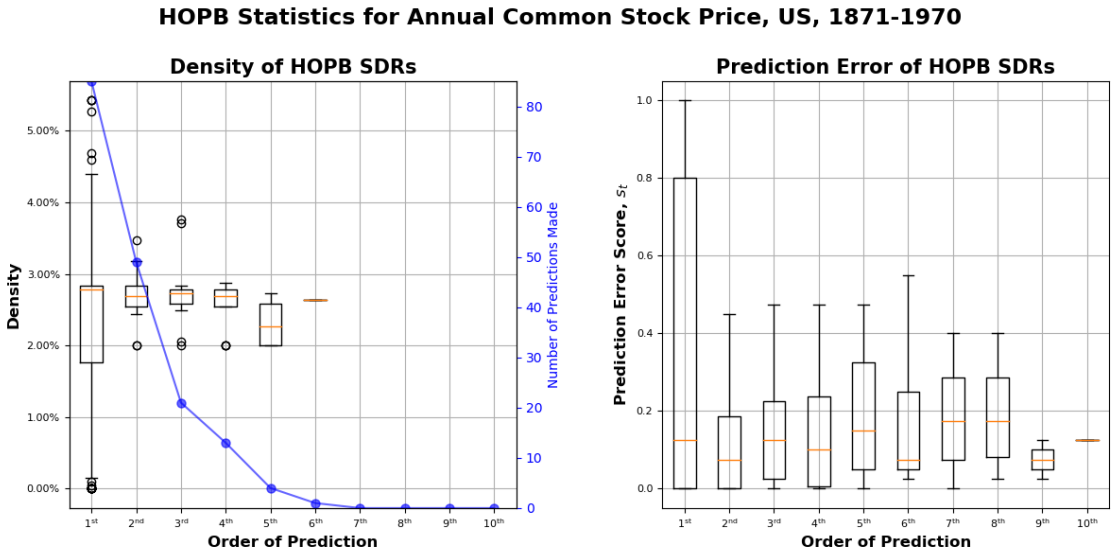

Figure 6.44: *HOPB statistics when using the annual common stock price dataset. Note that we are able to generate reliable and useful high-orders of prediction even with a dataset that is under 100 timesteps.*

6.6.5 Cybersecurity

As described in Section 5.2.3, included in this section are the results from the Smurf attack dataset which was retrieved from the KDD-Cup '99 Dataset [111]. We can visualize this dataset if Figure 6.45. We see the number of connections to the same host as the current connection rapidly increase when the Smurf attack begins. That is our target anomaly.

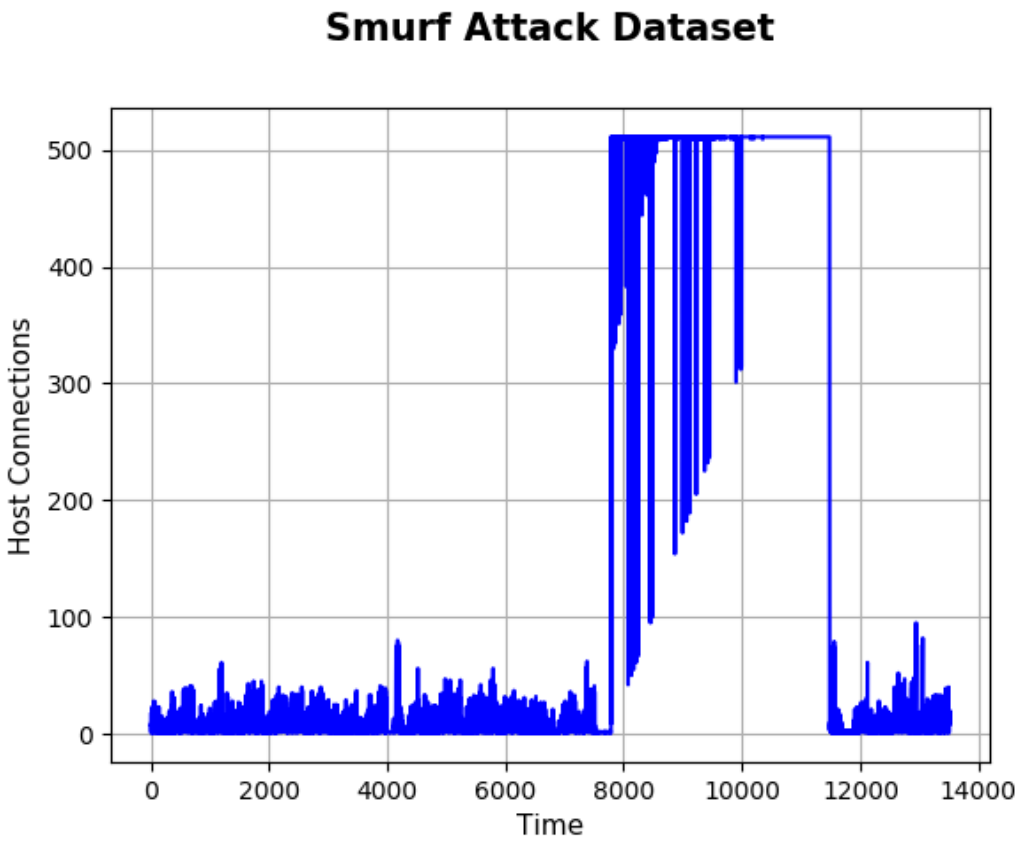

Figure 6.45: *Visualization of the Smurf attack dataset. The target anomaly occurs when the Smurf attack begins just before the* 8000^{th} *timestep which sends the number of connections to the same host as the current connection skyrocketing.*

The sample dataset in this example do not include any timestamps. So naturally there is no timestamp encoding taking place even if we wanted to do so. We just employ general purpose parameters for this dataset since we have enough data to work with. The exact parameters are $n_{\max} = 10$, $\psi = 500$, $\alpha = 0.8$ and $\epsilon = 0.01$. We first visualize the prediction errors over time in Figure 6.46.

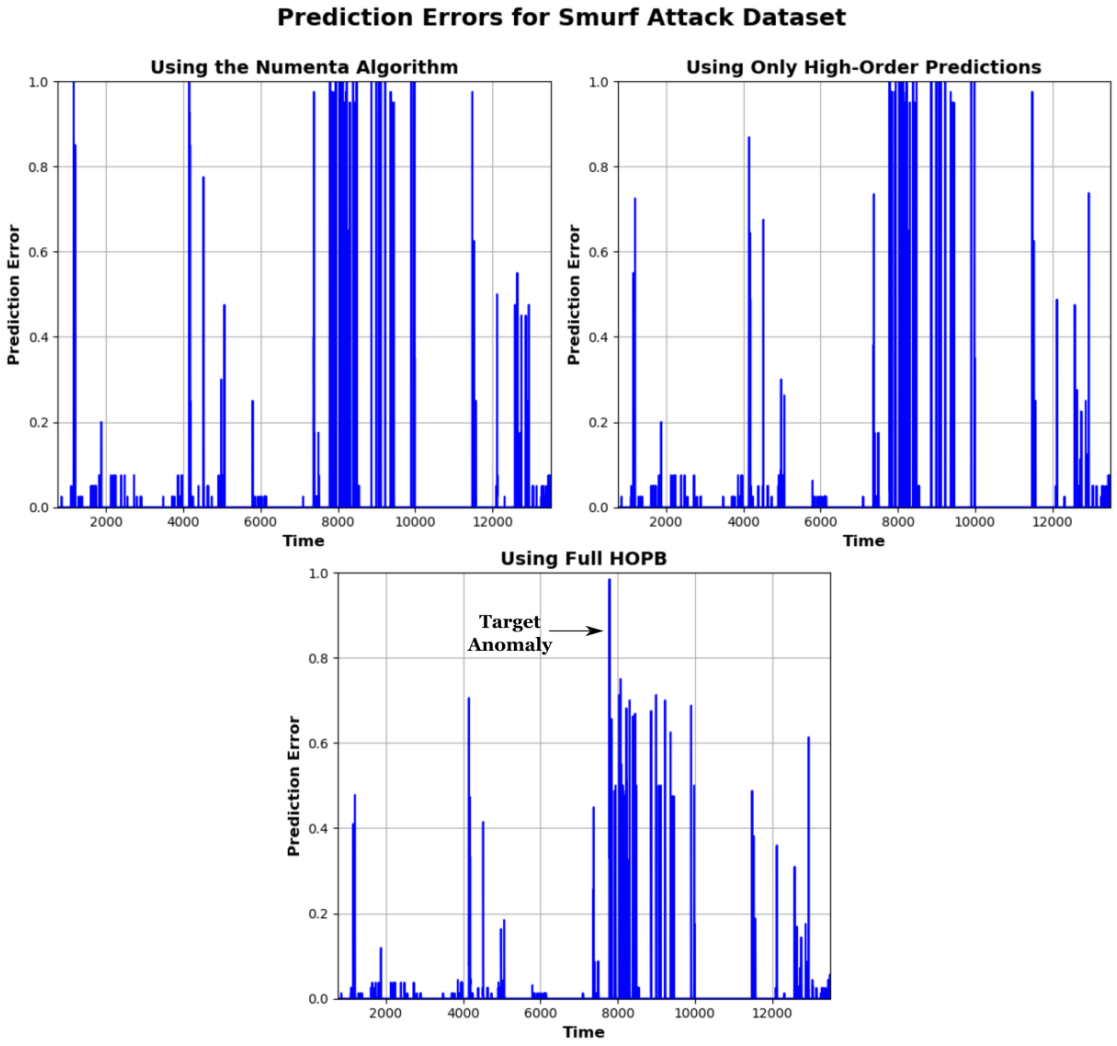

Figure 6.46: *Prediction errors over time for the Smurf attack dataset. Notice using only high-order predictions in the upper right plot allows us to see the difference between the most obvious anomaly and other lesser anomalous events. The bottom plot shows the full HOPB algorithm which illustrates the ability of the accumulating average to better isolate the target anomaly.*

In Figure 6.46, we see in the upper right plot that using high orders of prediction is better able to determine the most obvious anomalous event apart from the lesser anomalous events. You can match up the bursts of error in Figure 6.46 with sudden bursts of host connections in Figure 6.45 but those bursts are encompassed in normal network behavior. With respect to the level of prediction error, the Numenta algorithm is unable to discern between a normal network burst and an anomalous burst that corresponds to a Smurf attack. In the bottom plot when using the full HOPB algorithm we see a single

clean burst of maximum error exactly when the Smurf attack begins that is much more easily discernible from normal network behavior. We lastly visualize in Figure 6.47 how the number of predictions varies over time when using HOPB on this data. Notice an early burst of error brings the number of predictions down to first-order predictions but it quickly recovers until the Smurf attack occurs in which uncertainty is maximal and only first-order predictions are used.

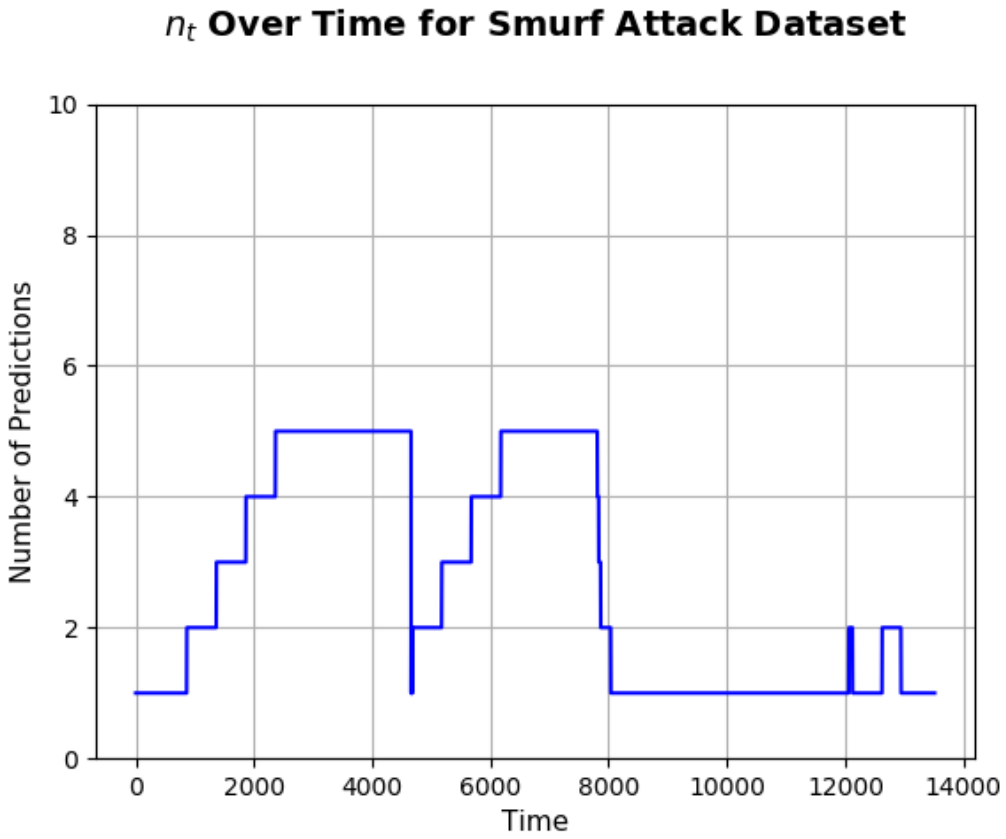

Figure 6.47: *Visualization of the number of predictions called for floating up and down when using HOPB on the Smurf attack dataset. An early burst of error brings the number of predictions down to first-order predictions but it quickly recovers until the Smurf attack occurs in which uncertainty is maximal and only first-order predictions are used.*

6.7 NAB Scores

In this section we report the normalized scores obtained from testing HOPB against NAB. A few different experimental setups with NAB are arranged for exploration purposes.

6.7.1 Top Scores

The comparative results with the other algorithms enumerated in Section 5.2.2.3 are shown in Table 6.1. These scores convey the *highest relative increase and best overall performance for HOPB compared to the Numenta algorithm* from varying the random initialization of the underlying HTM model. The scores reported for the Numenta algorithm are the scores discovered when using that same random initialization of the underlying HTM model. Concretely, if you wish to recreate these results, use a spatial pooler random seed of 6329 and a temporal memory random seed of 2814. In Section 6.7.2.2 we show that an increase in performance across all three profiles is stable across many random initializations with statistical significance. Furthermore, to get the results for HOPB and the Numenta algorithm, we used the latest available code and predetermined model parameters available in the NuPIC repository [120]. These predetermined parameters are recommended to be used by Numenta for this anomaly detection task. The only difference in the parameter settings for HOPB is that *we do not encode timestamps*. For the Numenta algorithm, we report the scores while encoding timestamps since it is apart of the original design. We show in Section 6.7.2.1 that encoding timestamps generates benefits performance for the Numenta algorithm but actually decreases the performance for HOPB. The general purpose HOPB parameters that obtained these results are $n_{\max} = 10$, $\psi = 500$, $\alpha = 0.8$ and $\epsilon = 0.01$.

Table 6.1: *Final highest NAB scores per application profile for each algorithm after parameter tuning.*

Algorithm	Standard	Low FN	Low FP
HOPB	72.08	75.64	67.32
CAD OSE	69.90	73.18	66.98
Numenta	68.37	73.20	61.27
KNN-CAD	57.99	64.81	43.41
Relative Entropy	54.64	58.84	47.60
Twitter ADVec	47.06	53.50	33.61
Windowed Gaussian	39.65	47.41	20.87
Etsy Skyline	35.69	44.48	27.08
Bayesian Online Changepoint Detection	17.71	32.26	3.21
EXPoSE	16.44	26.92	3.19
Random	11.00	19.50	1.20
Null	0.0	0.0	0.0

Note the relative improvement that HOPB provides compared to the Numenta algorithm. The largest increase appears to be in the low false positive profile jumping from approximately 61% to 67% which is a 6% increase. However, recall that many of the datasets used inside NAB are relatively short (between 1000 and 2000) timesteps. This rarely gives enough time for the underlying HTM model to establish meaningful second and third order lateral connections. Upon manual inspection, it appears that in the majority of the datasets the HOPB algorithm is rarely able to utilize any high-order predictions. The score increase is primarily due to the subset of datasets where these high-order predictions are able to take effect. For this reason, we feel the results suggest an underestimate of the true potential of HOPB.

6.7.2 Varying the Random Seeds

In this section, we explore the effects of varying the random seed to discover more about the algorithm. We test the effect of timestamp encoding on each algorithm separately and compare the best performances of both. We chose thirty pairs of random generated four digit seeds to test with.

6.7.2.1 Toggling Timestamp Encoding

In Section 6.2, there is mentioned a note about encoding timestamps with the underlying HTM model. We test both the Numenta algorithm and HOPB with the same parameters both with and without encoding timestamps to see the difference. Recall if the timestamps are not encoded, all the bits in the binary vector representation coming out of the encoder are used to represent the scalar input and all potentially useless or harmful timestamp information is discarded. The results for HOPB are shown in Table 6.2. The first set of random seeds are the seeds available by default in the NuPIC repository and are tuned to work best with the Numenta algorithm.

Table 6.2: *Results from toggling the encoding of timestamps and the impact that it has on the NAB scores across several random seeds when using HOPB. In this table, T corresponds to encoding the timestamp and F corresponds to not encoding the timestamp. T-F is the difference between the two for each pair result. All these results were discovered while using the general purpose HOPB parameters:* $n_{\max} = 10$, $\psi = 500$, $\alpha = 0.8$ *and* $\epsilon = 0.01$. *The starred random seeds are the default random seeds which were used to optimize the HTM parameters.*

Toggling Timestamp Encoding Using HOPB									
Random Seeds	**Standard**			**Low FN**			**Low FP**		
(SP, TM)	**T**	**F**	**T-F**	**T**	**F**	**T-F**	**T**	**F**	**T-F**
1956, 1960*	71.49	70.31	1.17	75.25	75.38	-0.13	66.41	65.36	1.05
3749, 8394	69.17	71.07	-1.90	73.41	75.54	-2.13	63.13	64.27	-1.14
1200, 2264	68.78	70.23	-1.45	72.87	74.69	-1.83	62.66	62.39	0.27
1439, 1624	70.55	70.2	0.35	74.91	74.96	-0.06	63.27	65.57	-2.30
1230, 2701	70.78	68.88	1.90	75.35	73.79	1.55	65.49	62.23	3.26
9432, 9388	68.22	68.90	-0.68	72.49	73.81	-1.32	61.24	61.79	-0.55
8271, 4267	70.43	70.14	0.29	75.12	74.92	0.20	62.38	63.2	-0.82
3628, 1152	70.38	70.38	0.00	74.79	74.8	0.00	64.76	64.02	0.74
4444, 9588	69.04	70.92	-1.88	73.33	75.15	-1.83	61.94	64.94	-3.00
6346, 1628	69.94	71.07	-1.12	74.79	75.54	-0.75	62.11	63.87	-1.75
5123, 6271	71.79	72.66	-0.86	76.31	77.46	-1.15	64.17	64.23	-0.06
8653, 2667	69.48	67.43	2.05	73.90	72.41	1.49	62.99	61.29	1.70
6329, 2814	68.86	72.08	-3.21	73.33	75.64	-2.31	62.95	67.32	-4.37
1623, 9582	69.22	71.58	-2.36	74.02	75.88	-1.86	62.87	64.73	-1.86
1994, 0178	70.03	69.11	0.92	74.85	73.74	1.11	61.57	63.48	-1.91
9182, 2983	68.13	71.55	-3.42	72.55	75.97	-3.42	61.34	65.91	-4.57
8819, 5178	70.66	71.54	-0.88	75.27	76.43	-1.16	62.62	64.2	-1.58
0062, 6125	67.49	69.63	-2.14	72.29	74.29	-2.00	60.56	62.43	-1.88
2998, 9238	68.04	69.86	-1.82	72.66	74.16	-1.50	61.8	62.89	-1.09
4165, 7096	68.70	70.96	-2.27	73.10	75.47	-2.37	61.51	63.81	-2.29
1649, 0421	70.26	71.73	-1.47	74.71	75.84	-1.13	63.01	65.38	-2.37
2077, 5906	70.80	71.12	-0.32	75.07	75.58	-0.50	65.08	63.93	1.15
7496, 3116	70.88	70.71	0.17	75.19	75.01	0.18	65.22	63.52	1.70
5888, 3807	67.84	69.29	-1.45	71.95	73.78	-1.83	62.46	62.74	-0.28
0178, 9158	68.44	69.04	-0.6	72.64	73.61	-0.98	62.93	61.18	1.75
3849, 2889	71.02	68.65	2.37	75.22	72.95	2.27	64.15	62.72	1.43
1374, 1806	69.58	69.76	-0.18	74.26	74.38	-0.12	61.9	62.99	-1.08
4976, 1848	70.43	71.68	-1.24	73.68	75.95	-2.27	65.71	64.86	0.85
6649, 1944	68.26	69.06	-0.8	71.94	73.62	-1.68	63.19	63.6	-0.41
1355, 4188	67.97	68.96	-0.99	72.04	73.56	-1.52	61.67	61.93	-0.26
Average	69.56	70.28	-0.73	73.91	74.81	-0.90	63.04	63.69	-0.66

Notice in Table 6.2 that encoding timestamps on average reduces the quality of the NAB scores across all three profiles. The results for the Numenta algorithm are shown in Table 6.3.

Table 6.3: *Results from toggling the encoding of timestamps and the impact that it has on the NAB scores across several random seeds when using the Numenta algorithm. In this table, T corresponds to encoding the timestamp and F corresponds to not encoding the timestamp. T-F is the difference between the two for each paired result. The starred random seeds are the default random seeds which were used to optimize the HTM parameters.*

Toggling Timestamp Encoding Using the Numenta Algorithm									
Random Seeds	**Standard**			**Low FN**			**Low FP**		
(SP, TM)	**T**	**F**	**T-F**	**T**	**F**	**T-F**	**T**	**F**	**T-F**
1956, 1960*	69.67	68.67	1.00	74.32	73.65	0.67	61.71	59.78	1.93
3749, 8394	69.42	70.08	-0.66	73.58	74.59	-1.01	63.47	63.38	0.09
1200, 2264	69.22	69.28	-0.06	73.73	73.77	-0.04	60.87	62.36	-1.49
1439, 1624	70.11	69.87	0.24	74.61	74.74	-0.13	62.90	61.67	1.23
1230, 2701	70.04	69.27	0.77	74.57	74.34	0.22	63.66	60.53	3.13
9432, 9388	67.97	69.27	-1.30	72.61	73.99	-1.37	61.98	61.98	0.00
8271, 4267	71.49	68.94	2.55	76.4	73.84	2.56	62.89	62.95	-0.06
3628, 1152	70.13	69.17	0.96	74.91	73.99	0.93	63.56	62.54	1.02
4444, 9588	68.59	69.18	-0.60	73.00	73.66	-0.66	60.34	61.78	-1.44
6346, 1628	70.85	68.82	2.03	75.4	73.75	1.64	63.38	61.75	1.63
5123, 6271	71.20	72.06	-0.87	75.91	76.78	-0.86	62.73	63.78	-1.05
8653, 2667	69.29	67.50	1.78	73.78	72.01	1.76	61.41	61.04	0.37
6329, 2814	68.37	70.59	-2.22	73.20	74.93	-1.73	61.27	63.02	-1.75
1623, 9582	70.85	69.87	0.98	75.39	74.74	0.65	62.99	61.84	1.15
1994, 0178	70.17	67.64	2.52	74.94	72.39	2.54	61.86	60.80	1.06
9182, 2983	67.24	69.85	-2.61	72.12	75.01	-2.89	58.42	60.69	-2.27
8819, 5178	69.63	71.95	-2.32	74.51	76.99	-2.49	64.29	63.07	1.22
0062, 6125	68.23	71.61	-3.38	72.79	76.19	-3.40	60.01	63.93	-3.92
2998, 9238	69.97	69.84	0.12	74.46	74.44	0.02	62.23	61.82	0.41
4165, 7096	70.01	70.67	-0.67	74.83	75.56	-0.73	62.71	62.19	0.52
1649, 0421	70.11	69.65	0.47	75.19	74.59	0.60	61.96	63.00	-1.04
2077, 5906	70.95	70.28	0.67	75.75	75.01	0.74	62.62	63.25	-0.63
7496, 3116	69.16	70.80	-1.63	73.72	75.36	-1.64	62.12	63.26	-1.15
5888, 3807	70.64	67.17	3.47	74.68	72.08	2.60	64.42	59.27	5.14
0178, 9158	69.3	68.59	0.71	74.07	73.60	0.47	61.76	62.36	-0.6
3849, 2889	70.4	69.17	1.22	75.38	74.56	0.82	62.23	61.58	0.66
1374, 1806	69.83	69.95	-0.12	74.71	74.51	0.21	61.23	61.94	-0.71
4976, 1848	69.3	69.68	-0.38	73.78	74.61	-0.83	60.97	62.51	-1.54
6649, 1944	68.88	69.76	-0.89	73.50	74.67	-1.17	60.66	60.98	-0.32
1355, 4188	68.61	70.36	-1.74	73.33	75.07	-1.74	60.19	62.25	-2.06
Average	69.65	69.65	0.00	74.31	74.45	-0.14	62.03	62.04	-0.02

Notice in Table 6.3 that encoding timestamps on average produces approximately the same results in all three profiles. The combined results from Table 6.3 and Table 6.2 suggest that using HOPB replaces the need for encoding timestamps for anomaly detection. Capturing context is accomplished in HOPB by extracting it from the high-order transitions stored in the network itself. This is a much more autonomous and general solution than manually encoding the timestamps themselves.

6.7.2.2 HOPB versus Numenta

In this section, we compare the Numenta algorithm performance and HOPB performance on NAB while encoding timestamps and not encoding timestamps respectively. To be clear, these are the scores when not using the HOPB framework (Numenta algorithm) and using the HOPB framework (HOPB algorithm) on top of the same HTM model. Encoding timestamps is apart of the original design for the Numenta algorithm and we show it doesn't significantly effect the results in Table 6.3. The comparative results are displayed in Table 6.4.

Table 6.4: *Comparing the performance of HOPB and the Numenta algorithm while varying the random seeds. In this table, H corresponds to using HOPB and N corresponds to using the Numenta algorithm. H-N corresponds to the difference between the paired scores. The same general purpose HOPB parameters ($n_{\max} = 10$, $\psi = 500$, $\alpha = 0.8$ and $\epsilon = 0.01$) were used for each random seed pair. The highlighted cells show the maximum scores obtained for each algorithm for each profile. The starred random seeds are the default random seeds which were used to optimize the HTM parameters.*

HOPB Versus Numenta									
Random Seeds	**Standard**			**Low FN**			**Low FP**		
(SP, TM)	**H**	**N**	**H-N**	**H**	**N**	**H-N**	**H**	**N**	**H-N**
1956, 1960*	70.31	69.67	0.64	75.38	74.32	1.06	65.36	61.71	3.65
3749, 8394	71.07	69.42	1.65	75.54	73.58	1.96	64.27	63.47	0.80
1200, 2264	70.23	69.22	1.01	74.69	73.73	0.96	62.39	60.87	1.52
1439, 1624	70.2	70.11	0.09	74.96	74.61	0.35	65.57	62.9	2.67
1230, 2701	68.88	70.04	-1.16	73.79	74.57	-0.77	62.23	63.66	-1.42
9432, 9388	68.9	67.97	0.93	73.81	72.61	1.20	61.79	61.98	-0.20
8271, 4267	70.14	71.49	-1.35	74.92	76.40	-1.48	63.2	62.89	0.31
3628, 1152	70.38	70.13	0.25	74.8	74.91	-0.12	64.02	63.56	0.46
4444, 9588	70.92	68.59	2.33	75.15	73.00	2.16	64.94	60.34	4.60
6346, 1628	71.07	70.85	0.21	75.54	75.40	0.14	63.87	63.38	0.49
5123, 6271	72.66	71.20	1.46	77.46	75.91	1.55	64.23	62.73	1.50
8653, 2667	67.43	69.29	-1.86	72.41	73.78	-1.37	61.29	61.41	-0.12
6329, 2814	72.08	68.37	3.71	75.64	73.20	2.44	67.32	61.27	6.05
1623, 9582	71.58	70.85	0.73	75.88	75.39	0.48	64.73	62.99	1.74
1994, 0178	69.11	70.17	-1.05	73.74	74.94	-1.20	63.48	61.86	1.62
9182, 2983	71.55	67.24	4.31	75.97	72.12	3.84	65.91	58.42	7.49
8819, 5178	71.54	69.63	1.90	76.43	74.51	1.92	64.20	64.29	-0.09
0062, 6125	69.63	68.23	1.39	74.29	72.79	1.50	62.43	60.01	2.42
2998, 9238	69.86	69.97	-0.11	74.16	74.46	-0.30	62.89	62.23	0.66
4165, 7096	70.96	70.01	0.96	75.47	74.83	0.64	63.81	62.71	1.09
1649, 0421	71.73	70.11	1.62	75.84	75.19	0.65	65.38	61.96	3.42
2077, 5906	71.12	70.95	0.17	75.58	75.75	-0.17	63.93	62.62	1.31
7496, 3116	70.71	69.16	1.55	75.01	73.72	1.30	63.52	62.12	1.41
5888, 3807	69.29	70.64	-1.35	73.78	74.68	-0.90	62.74	64.42	-1.68
0178, 9158	69.04	69.30	-0.26	73.61	74.07	-0.46	61.18	61.76	-0.58
3849, 2889	68.65	70.40	-1.75	72.95	75.38	-2.43	62.72	62.23	0.49
1374, 1806	69.76	69.83	-0.07	74.38	74.71	-0.33	62.99	61.23	1.76
4976, 1848	71.68	69.30	2.38	75.95	73.78	2.16	64.86	60.97	3.89
6649, 1944	69.06	68.88	0.18	73.62	73.5	0.12	63.60	60.66	2.94
1355, 4188	68.96	68.61	0.35	73.56	73.33	0.23	61.93	60.19	1.74
Average	70.28	69.65	0.63	74.81	74.31	0.50	63.69	62.03	1.66

In Table 6.4, we see that HOPB consistently performs better across many different random initializations. Tuning the HOPB parameters for each random seed pair would likely result in the kind of performance increase seen in Table 6.1. The important piece to takeaway from this analysis is the fact that HOPB on average performs better across many different random initializations even with constant parameters. As suggested in Section 6.7.2.1, if we compare HOPB to the Numenta algorithm while not encoding timestamps on each algorithm, the performance gap is wider still. The strongest increase in performance seems to be from the reward low false positive profile presumably from the fault tolerance aspect of the algorithm. The datasets inside NAB are likely not long enough for good high-order predictions to be established which gives the algorithm better discernibility to complex anomaly models. Examples where this latter effect occurs can be seen throughout the rest of this chapter.

Paired t-Test We can use a paired t-test to help back up the claim made in this section that HOPB provides a statistically significant performance improvement across all three application profiles. Concretely, we can perform three paired t-tests (one for each application profile) using the data created by varying the random seed.

Define the mean score of using HOPB and the Numenta algorithm as μ_{H} and μ_{N} respectively. The null hypothesis, H_0, is defined as $\mu_{\text{H}} = \mu_{\text{N}}$. The alternative hypothesis, H_a, is defined as $\mu_{\text{H}} > \mu_{\text{N}}$. This corresponds to a right-tailed test, for which a t-test for paired samples can be used. Since there are 30 samples for each application profile, we have a degrees of freedom of 29. We set the significance level to 0.03. Thus, our rejection region is defined as $\mathcal{R} = \{t : t > 1.957\}$. The results for each application profile are shown below.

1. **Standard:**

   The sample t-statistic under the standard application profile, t_{s}, is calculated below.

   $$t_{\text{s}} = \frac{\bar{D}}{s_D/\sqrt{n}} = \frac{0.629}{1.483/\sqrt{30}} = 2.324$$

   In conclusion, since $t_{\text{s}} \in \mathcal{R}$, we reject H_0. There is enough evidence to claim that population mean μ_{H} is greater than μ_{N} at the 0.03 significance level.

2. **Low FN:**

   The sample t-statistic under the low false negative application profile, t_{fn}, is calculated below.

$$t_{\text{fn}} = \frac{\bar{D}}{s_D/\sqrt{n}} = \frac{0.504}{1.368/\sqrt{30}} = 2.017$$

In conclusion, since $t_{\text{fn}} \in \mathcal{R}$, we reject H_0. There is enough evidence to claim that population mean μ_{H} is greater than μ_{N} at the 0.03 significance level.

3. **Low FP:**

   The sample t-statistic under the low false positive application profile, t_{fp}, is calculated below.

$$t_{\text{fp}} = \frac{\bar{D}}{s_D/\sqrt{n}} = \frac{1.665}{2.041/\sqrt{30}} = 4.468$$

   In conclusion, since $t_{\text{fp}} \in \mathcal{R}$, we reject H_0. There is enough evidence to claim that population mean μ_{H} is greater than μ_{N} at the 0.03 significance level.

In the above hypothesis tests, one can see the most evidence of improvement by far is within the low false positive application profile. This is likely do to the effect of increased fault tolerance that HOPB provides. This prevents us from accidentally tagging several false positives.

6.8 Latency

In this section we analyze the time complexity of HOPB versus the Numenta algorithm. An increase in detection capabilities is relatively meaningless in streaming scenarios if the algorithm cannot run in real-time. The HOPB algorithm adds some extra steps to the Numenta algorithm so we are interested in the additional time it takes to perform these steps. In theory, the maximum cost should occur when HOPB is making the maximum number of predictions at each timestep. For these experiments, we assume the general purpose maximum of $n_{\max} = 10$ which is used throughout this thesis.

We empirically measure the execution time for the Numenta algorithm and HOPB under two different dataset scenarios. In one setting, we feed in 20,000 timesteps of a perfectly predicable sine wave. We measure the time it takes to fully process each record from initializing the models to processing the final timestep. In the second scenario we feed in 20,000 timesteps of sinusoidal data that has been perturbed with uniformly distributed noise that has a maximum amplitude of 20% of the data value range. Each reported time is an average execution time over 20 executions where a brand new randomly initialized

model is used each iteration. All of the following execution times were captured using a standard laptop environment with an Intel Core i7 7700HQ processor.

The results of the first scenario with 20,000 timesteps of a perfect sine wave is shown in Table 6.5. In the table, "Force_Max" means we are forcing that many predictions to be generated at each timestep, if they can be. The last row details the average time taken for using HOPB with reasonable parameters.

Table 6.5: *Latency results when measuring the average execution time of 20,000 timesteps of a perfect sine wave.*

Latency on a Perfect Sine Wave

Algorithm	Average (s)	Seconds/Record
Numenta	151.97	0.0076
HOPB - Force_Max, n_max=1	80.45	0.0040
HOPB - Force_Max, n_max=2	90.45	0.0045
HOPB - Force_Max, n_max=3	98.96	0.0049
HOPB - Force_Max, n_max=4	107.88	0.0054
HOPB - Force_Max, n_max=5	115.8	0.0058
HOPB - Force_Max, n_max=6	124.46	0.0062
HOPB - Force_Max, n_max=7	132.51	0.0066
HOPB - Force_Max, n_max=8	139.84	0.0070
HOPB - Force_Max, n_max=9	147.74	0.0074
HOPB - Force_Max, n_max=10	154.74	0.0077
HOPB - n_max=10, psi=1000, epsilon=0.02, alpha=0.8	91.66	0.0046

In Table 6.5, notice that forcing only one order of prediction at each timestep is significantly faster than the Numenta algorithm despite the fact that they are executing the same algorithm short of the accumulating average mechanism. There are several optimizations and software simplifications used in our code that cause in the speedup. More interestingly, notice that despite forcing ten times the number of predictions at each timestep, we only approximately double the execution time. When using reasonable parameters, note we are able to process a new record every 0.0045 seconds. This is viable for virtually all real-time applications. We plot the average execution times in Figure 6.48 and see an almost perfect linear growth.

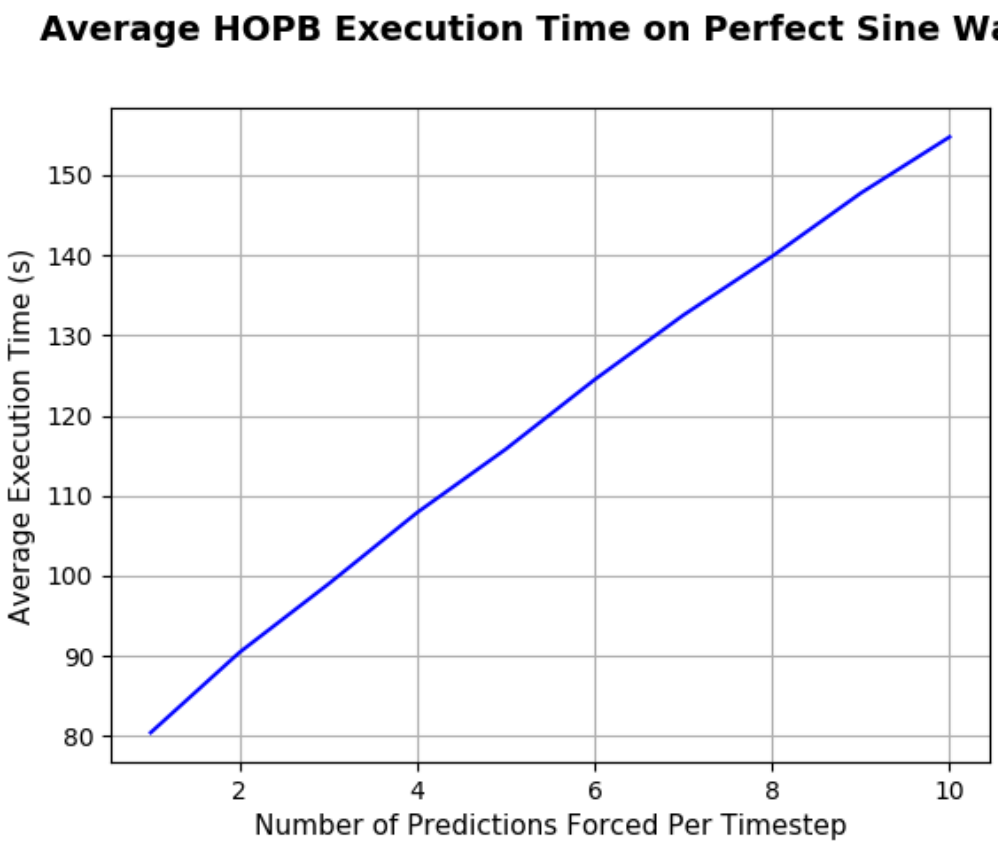

Figure 6.48: *Visualization of how the latency increases when using higher orders of prediction on a perfect sine wave. Notice the almost perfect linear growth in average execution time.*

We then add uniform noise to the sine wave with an amplitude of 20% of the total range of the data values. The results of this second scenario are displayed in Table 6.6.

Table 6.6: *Latency results when measuring the average execution time of 20,000 timesteps of a sine wave perturbed with uniform noise that has a maximum amplitude of 20% of the data value range.*

Latency on Sine with 20% Amplitude Uniform Noise

Algorithm	Average (s)	Seconds/Record
Numenta	249.79	0.0125
HOPB - Force_Max, $n_{\max} = 1$	174.31	0.0087
HOPB - Force_Max, $n_{\max} = 2$	193.09	0.0097
HOPB - Force_Max, $n_{\max} = 3$	210.33	0.0105
HOPB - Force_Max, $n_{\max} = 4$	227.96	0.0114
HOPB - Force_Max, $n_{\max} = 5$	245.49	0.0123
HOPB - Force_Max, $n_{\max} = 6$	261.51	0.0131
HOPB - Force_Max, $n_{\max} = 7$	278.90	0.0139
HOPB - Force_Max, $n_{\max} = 8$	293.48	0.0147
HOPB - Force_Max, $n_{\max} = 9$	308.46	0.0154
HOPB - Force_Max, $n_{\max} = 10$	325.05	0.0163
HOPB - $n_{\max} = 10$, $\psi = 1000$, $\epsilon = 0.02$, $\alpha = 0.8$	197.98	0.0099

In Table 6.6, notice that the execution times are significantly larger compared to the times in Table 6.5. When the sine wave is significantly perturbed with noise, the level of uncertainty in the data increases. More uncertainty in the data results in denser SDRs and more bursting events in the underlying HTM network. This increases the amount of computation needed at each execution of the temporal memory algorithm and thus a slowdown occurs. It seems that with reasonable parameters, HOPB is able to comfortably process each record in 0.01 seconds which is double the time when the sine wave included no noise but this is still viable for many real-time applications. Figure 6.49 plots the execution times as a function of $n_{\max}$ and we once again see almost perfect linear growth.

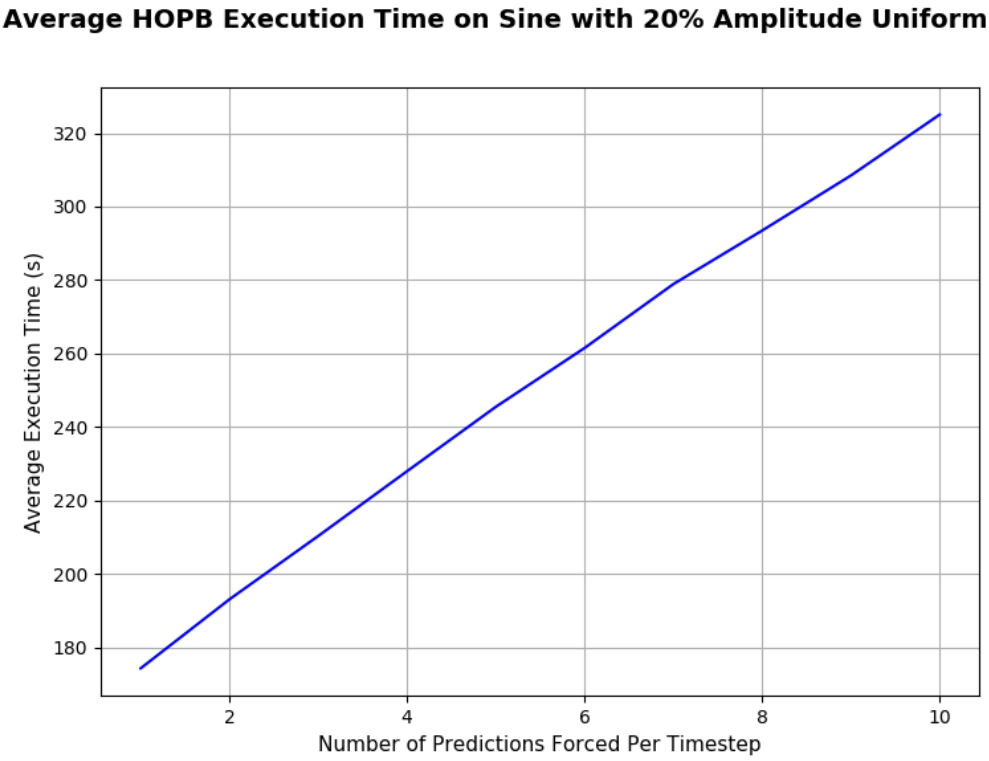

Figure 6.49: *Visualization of how the latency increases when using higher orders of prediction on a sine wave perturbed with uniform noise that has a maximum amplitude of 20% the range of values. Notice the almost perfect linear growth in average execution time.*

We wish to obtain a theoretical understanding of the increased time complexity of using HOPB on top of a time-series modeling algorithm. Figures 6.48 and 6.49 suggest the time complexity of using HOPB in the way described in this thesis is bounded above as in Equation 6.1. We say bounded above because when using a dynamic chain size the number of predictions actually being generated is often less than $n_{\max}$. We denote the time complexity of generating first-order predictions in the underlying time-series modeling algorithm $\mathcal{T}$ and the time complexity of using HOPB on top of that algorithm as $\mathcal{H}$.

$$\mathcal{H} = O\left(\left(1 + \frac{n_{\max} - 1}{\gamma}\right)\mathcal{T}\right) \quad \text{where} \quad n_{\max} \geq 1, \gamma > 0 \tag{6.1}$$

Where γ is a constant scalar that is characteristic of the underlying time-series modeling algorithm. In our case, when using HTM, one can see it appears $\gamma \approx 9$. Note that $n_{\max}$ is always a constant through the course of execution. Thus, when analyzing the time complexity in the asymptote of $\mathcal{T}$, we obtain the following in Equation 6.2.

$$\lim_{\mathcal{T}\to\infty} \frac{\mathcal{H}}{\mathcal{T}} = \lim_{\mathcal{T}\to\infty} \frac{\left(1 + \frac{n_{\max}-1}{\gamma}\right)\mathcal{T}}{\mathcal{T}} = 1 + \frac{n_{\max}-1}{\gamma} \ ,$$

$$0 < 1 + \frac{n_{\max}-1}{\gamma} < \infty \ \text{ where } \ n_{\max} \geq 1, \gamma > 0 \Longrightarrow \mathcal{H} = \Theta(\mathcal{T}) \tag{6.2}$$

Thus, we see in Equation 6.2 that the time complexity of using the HOPB framework possesses the same asymptotic behavior as the underlying time-series algorithm regardless of the HOPB parameter settings. Intuitively this makes sense as the number of additional predictions made at each timestep is always limited to a relatively small number.

Chapter 7

Conclusions

This section will conclude the thesis including a summary of the entire document, some deployment advice for readers interested in using HOPB and discussion about possible extensions and future work.

7.1 Thesis Summary

Presented in this thesis is a novel framework for streaming anomaly detection applicable to any time-series modeling algorithm capable of making high-order predictions. As a case example, we've formalized and demonstrated the framework's benefit on top of a Hierarchal Temporal Memory (HTM) network which is particularly well suited for streaming environments due to its unsupervised and continuous learning properties. The framework does not forfeit those strengths in any capacity.

To accomplish this, we formalize a "predictions on predictions" technique to obtain high-order predictions of the state of cellular activation within the HTM network at any point in time. In the framework, we accumulate multiple predictions made at different points in the past for each timestep individually to maintain immediate, real-time processing of timesteps without the need for any lookahead or tagging past timesteps. Apart of the framework is a custom data structure to efficiently manage multiple overlapping chains of prediction in real-time. Also, a method of dynamically choosing the appropriate number of predictions to be made per timestep that monitors specific prediction quality related properties of HTM is introduced. We formalize a degree of confidence that consolidates the error among multiple predictions for each timestep to get one final measure of prediction error that is contextually aware and fault tolerant. Since this framework is designed to distribute the final prediction error across multiple timesteps that are anoma-

lous with respect to local contextual patterns, we introduce an accumulating average of prediction error that helps isolate only the anomalous points in time that have the most supporting evidence to be so. Lastly, similar to the work in [77] and [12], we model the resulting values of the accumulating average as a rolling normal distribution to derive a likelihood that any pattern of error is truly anomalous.

We show in this thesis that the proposed framework adds an effective layer of fault tolerance as well as an increased ability to detect complex temporal anomalies with HTM. The effect of using HOPB can be thought of in some sense as a similar effect of using an ensemble technique as we pool together multiple predictions for each timestep made at different points in time. We achieve this "ensemble" effect with only once instance of a model, however. We demonstrate that the proposed framework redefines state-of-the-art performance on a popular streaming anomaly benchmark known as the Numenta Anomaly Benchmark. We provide additional evidence of this performance benefit from five real-world, third-party datasets that originate from a variety of domains where streaming anomaly detection can have a meaningful impact. Lastly, we empirically measure and report the execution times of the proposed framework with HTM and demonstrate evidence that it does not significantly alter the asymptotic time complexity of the underlying time-series modeling algorithm. The speed of the proposed framework as implemented in the Python programming language is shown to be appropriate for nearly all real-time applications while running on modest hardware.

Given the results of this thesis, we accept the hypothesis as it is stated in Section 5.1.

7.2 Deployment Strategies & Advice

Readers interested in using HOPB may be interested to know where to begin, how to choose the HOPB parameters and what to expect in the long-term. Setting up the HOPB framework should be relatively straightforward given the thorough descriptions of the necessary data structures and algorithms in Chapter 4. An example implementation of the HOPB framework with HTM as the underlying time-series modeling algorithm was used for the experiments in this thesis. Obviously, certain modifications will need to be made if you desire to use a different underlying time-series modeling algorithm. Concretely, the user will need to define a measure of prediction error for each "actual" versus "predicted" comparison and a measure of overall order of prediction quality that makes sense for your model. When using HTM, prediction error is defined as the normalized number of overlapping bits between the actual and predicted *internal* state of the network. However, if your model produces predictions in the same target format as your time-series data,

for instance, you may wish to define the distance between data values instead to capture prediction error. In terms of defining overall order of prediction quality when using HTM in the way that is defined in this thesis, it turns out that SDR density plays a key role in determining prediction quality as well as the past statistics of prediction error. It may very well be with other methods of generating high-order predictions and other models altogether that only the past statistics of prediction error are of interest. This would eliminate the need for $d_{\min}$ and $d_{\max}$ in Algorithm 3. Additionally, when using HTM the way that is defined in this thesis, it turns out that the frequency of acceptable prediction generation is an important factor to monitor. This may not be true for other methods of generating high-order predictions which would eliminate the need for α in Algorithm 2. The high-level HOPB framework is agnostic to how these metrics are defined.

Assuming the use of HTM as the underlying time-series modeling algorithm and using the HOPB framework as it is formalized in this thesis, the process is fairly autonomous. If you have access to a historical record of streaming data that is representative of the signal you wish to monitor it is recommended to experiment with HTM and HOPB parameters to determine optimal settings. The general purpose parameters for both HTM and HOPB should be used as a starting point. We demonstrate in this thesis that HTM networks need very little data to reliably learn signal patterns (compared to deep learning for instance) but processing larger amounts of widely representative normal signal patterns through the HTM network will still benefit performance. This is especially true with HOPB since high-order predictions require the existence of a relatively large amount of complex lateral connections. HTM networks are designed to continuously learn thus longterm upkeep is minimal.

7.2.1 Importance of Encoder Design

For using HOPB with HTM specifically, it cannot be stressed enough the importance of effective encoder design. The encoder is the engineer's entry point to explicitly define the underlying semantics of your data. This is like explicitly telling the HTM network what you want it to learn. Recall from a biological perspective encoders play the role of sensory organs which are responsible for reading external stimuli from the outside world which is later processed in the brain. Animal brains never receive any kind of stimuli directly from the outside world. For instance, the cochlea is an example of a biological encoder which encodes the frequency and amplitude of sound waves into a sparse pattern of neuron activity using the stimulation of tiny hairs [126]. The HTM encoder is the only piece of the algorithm that has any communication with the outside world (the time-series data). As the first step in the process, a faulty encoder will bottleneck the performance of the

rest of the entire system.

Throughout this thesis, we exclusively consider the encoding of simple scalar values. This is a relatively simple encoder design that is only meant to capture the distances between scalar values as their underlying semantics. This is done to provide a sense of comparability between past work and all experiments in this thesis. We are in no way limited to encoding only scalar values, however. Humans are relatively limited in their ability to encode only a preset range of stimuli from the outside world such as sound within a frequency range and visible light. It's hard coded into our DNA and relatively unchangeable. With artificial learning systems, however, we have the freedom to build whatever artificial sensory encoders we can imagine. We need only obey the encoder design rules brought fourth in [103] and discussed in Section 3.1.3. For instance, one may wish to monitor several features at once including a mixture of scalar and nominal values. These features could represent anything. Your time-series data may not even arrive in the form of relational tuples of features such as when working with images or packet capture (PCAP) files when monitoring computer networks. Discussed more in-depth in Section 3.1.3, the data need only be brought into a binary vector representation where overlapping bits imply semantic similarity. There can be many different possible encoder designs for a given data signal format. Intelligently designing and evaluating different encoder designs for various signal formats is a hotly researched topic in the HTM community.

7.3 Limitations & Future Work

In this section, we will discuss some of the current limitations of using HOPB with HTM and some possible ways to overcome them. New directions and extensions to this work are also discussed. The concern detailed in this section is related to using HTM specifically and not HOPB itself.

7.3.1 Alternative Methods For Getting High-Order Predictions

The procedure described in Section **??** requires a very robust and accurate population of lateral connections in the HTM network to work well. Generating high-order predictions out of HTM this way is somewhat unreliable and unwieldy. In general, reliable high-order transitions are established only after many reliable first-order transitions are learned which is often difficult to do in the first place. Furthermore, lateral connections are not remotely guaranteed to produce SDRs with a reasonable density and thus may not reflect a reasonable prediction. These facts likely severely limit the potential of using HOPB with HTM. Also, in reality, the HTM model is not directly learning high-order transitions

by design. Any high-order transition we use was learned as a first-order transition from a different state and context. This sometimes leads to a translation effect where a high-order transition is very similar, if not equivalent, to a first-order transition made later. Ideally, the predictions of high-order transitions should be made as independently as possible to capitalize on the "ensemble" effect.

7.3.1.1 Separate Sets of Dendritic Branches

To mitigate this, one may consider establishing separate temporal memory regions (sets of lateral dendritic connections) for each order of prediction you plan to use. These sets of dendritic branches could share one encoder and one spatial pooler but would learn their own separate lateral connections. Each separate set of dendritic branches would be directly learning to predict its assigned order of prediction and only that. Note for high-orders of prediction, synaptic weight updates would be slightly delayed by design. For a given set of dendritic branches intended to predict the column activations i timesteps into the future, we wouldn't update the lateral synapse permanences based on the predicted state until we have the spatial pooler activations for timestep $t + i$. This ensures only i^{th}-order transitions are learned in that region. At each timestep, we would store the predicted state arising from the spatial pooler activations for that timestep for each order of prediction anyway since they would be needed for computing average prediction error. The rest of the HOPB framework could remain the same way it's designed.

Pros Each order of prediction is learning its own dedicated set of lateral connections. The high-order predictions obtained from these separate models would always be at an acceptable density and likely be much more accurate. We are essentially only using the temporal memory algorithm how its designed to be used here. Additionally, each temporal memory region is learning completely independently of the rest which would truly capitalize on the "ensemble" effect on fault tolerance shown in Section 6.5.4.

Cons Note that the computational costs, both time and memory, of maintaining multiple sets of dendritic branches at once would be great. However, the natural parallelism of the situation could likely significantly reduce the time complexity. Each set of lateral connections is independent of the rest. Also, dynamically choosing the number of predictions would become an arduous process since entire sets of dendritic branches would need to be created *and* trained or destroyed to float the number of predictions up and down. We wouldn't even be able to start collecting data about how an order of prediction is performing until after it has been successfully trained which is not easy to determine

when that has happened or ever will happen in general. In a benchmark evaluation such as NAB, this number would likely somehow have to be determined prior to reading any data since the datasets are so small.

7.3.1.2 High-Order Synapse Updates

A separate methodology is to try to encourage a single HTM network to learn high-order transitions. This is to alter the learning procedure in temporal memory to include rewarding synapses that led to correct high-order predictions. We obtain a set of multiple predictions made at each timestep with HOPB. Currently, the only learning that takes place is from the result of first-order predictions. The high-order predictions are used to calculate prediction error for timesteps, but the accuracy of those predictions do not alter the synapse permanences of the model in any way. It would make sense to reward lateral connections that led to correct high-order predictions in addition to rewarding lateral connections that led to correct first-order predictions. This would pose the learning objective of HTM model to be able to predict the cellular activity of the short-term future instead of just the next timestep exclusively. From the perspective of nature, we obviously do not live in a universe that is composed of discrete timesteps. We experience time continuously. Discretization of time allows us to mathematically model and implement algorithms inspired by the neocortex in digital computers. However, there is no biological argument for the strict adherence to discrete intervals of time. It does not forfeit biological plausibility to blur the discretization of time and predict and learn using the short-term future instead of strictly single timestep transitions. It would be interesting to see if this kind of learning paradigm might help HTM models learn sequences in general as more useful and complex lateral connections would naturally be established.

7.3.2 Using Different Underlying Time-Series Models

An obvious extension to this work is to develop different formalizations of the HOPB framework with different underlying time-series modeling algorithms. The results presented in this thesis suggest a considerable performance benefit can be expected. This could include using the HOPB framework on top of Long-Short Memory Networks or Hidden Markov Models, for instance. The only requirement of the time-series model is that it is able to produce high-order predictions. If that is not natural in the model design, one may consider developing separate models for each order of prediction similar to what is discussed in Section 7.3.1.1. Testing HOPB with many different underlying time-series modeling algorithms would help solidify the benefit it provides to streaming

anomaly detection in general.

Bibliography

[1] Brain research through advancing innovative neurotechnologies (brain). URL https://braininitiative.nih.gov/.

[2] Rethinking the brain. URL https://www.nature.com/news/rethinking-the-brain-1.17168.

[3] URL http://www.opencyc.org/.

[4] URL http://www.aics.riken.jp/en/.

[5] Jeopardy! (series). URL http://tvtropes.org/pmwiki/pmwiki.php/Series/Jeopardy.

[6] *American standard code for information interchange. Sponsor: Business Equipment Manufacturers Association. Approved June 17, 1963.* 1963.

[7] etsy/skyline, October 2015. URL https://github.com/etsy/skyline.

[8] Examples of electromyograms, Oct 2016. URL https://www.physionet.org/physiobank/database/emgdb/.

[9] Artificial intelligence pioneer says we need to start over, Sep 2017. URL https://www.axios.com/ai-pioneer-advocates-starting-over-2485537027.html.

[10] Ryan P. Adams and David J.C. MacKay. Bayesian online changepoint detection. Cambridge, UK, 2007.

[11] Subutai Ahmad and Jeff Hawkins. Properties of sparse distributed representations and their application to hierarchical temporal memory. *CoRR*, abs/1503.07469, 2015. URL http://arxiv.org/abs/1503.07469.

[12] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134147, 2017. doi: 10.1016/j.neucom.2017.04.070.

[13] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md. Rafiqul Islam. A survey of anomaly detection techniques in financial domain. *Future Gener. Comput. Syst.*, 55 (C):278–288, February 2016. ISSN 0167-739X. doi: 10.1016/j.future.2015.01.001. URL https://doi.org/10.1016/j.future.2015.01.001.

[14] Dario Antonelli, Giulia Bruno, and Silvia Chiusano. Anomaly detection in medical treatment to discover unusual patient management. *IIE Transactions on Healthcare Systems Engineering*, 3(2):69–77, 2013. doi: 10.1080/19488300.2013.787564. URL http://dx.doi.org/10.1080/19488300.2013.787564.

[15] Loïc Bontemps, Van Loi Cao, James McDermott, and Nhien-An Le-Khac. Collective anomaly detection based on long short term memory recurrent neural network. *CoRR*, abs/1703.09752, 2017. URL http://arxiv.org/abs/1703.09752.

[16] E. Burnaev and V. Ishimtsev. Conformalized density- and distance-based anomaly detection in time-series data. *ArXiv e-prints*, August 2016.

[17] W.H. Calvin and inc Scientific American. *Emergence of Intelligence*. Scientific American, Incorporated, 1998. ISBN 9781591763642. URL https://books.google.com/books?id=fonnjwEACAAJ.

[18] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *J. ACM*, 58(3):11:1–11:37, June 2011. ISSN 0004-5411. doi: 10.1145/1970392.1970395. URL http://doi.acm.org/10.1145/1970392.1970395.

[19] Tony F. Chan, Gene H. Golub, and Randall J. Leveque. Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician*, 37(3):242–247, 1983. doi: 10.1080/00031305.1983.10483115. URL https://www.tandfonline.com/doi/abs/10.1080/00031305.1983.10483115.

[20] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL http://doi.acm.org/10.1145/1541880.1541882.

[21] Deepthi Cheboli. *Anomaly detection of time series*. PhD thesis, 2010.

[22] Haibin Cheng, Pang Ning Tan, Christopher Potter, and Steven Klooster. *Detection and characterization of anomalies in multivariate time series*, volume 1, pages 409–420. 2009. ISBN 9781615671090.

[23] Yoonsuck Choe. *Hebbian Learning*, pages 1305–1309. Springer New York, New York, NY, 2015. ISBN 978-1-4614-6675-8. doi: 10.1007/978-1-4614-6675-8_672. URL https://doi.org/10.1007/978-1-4614-6675-8_672.

[24] David Cole. The chinese room argument, Mar 2004. URL https://plato.stanford.edu/entries/chinese-room/.

[25] B. S. J. Costa, C. G. Bezerra, L. A. Guedes, and P. P. Angelov. Online fault detection based on typicality and eccentricity data analytics. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, July 2015. doi: 10.1109/IJCNN.2015.7280712.

[26] Yuwei Cui, Chetan Surpur, Subutai Ahmad, and Jeff Hawkins. Continuous online sequence learning with an unsupervised neural network model. *CoRR*, abs/1512.05463, 2015. URL http://arxiv.org/abs/1512.05463.

[27] Yuwei Cui, Subutai Ahmad, and Jeff Hawkins. The htm spatial pooler: a neocortical algorithm for online sparse distributed coding. *bioRxiv*, 2016. doi: 10.1101/085035. URL https://www.biorxiv.org/content/early/2016/11/02/085035.

[28] Srdjan D Antic, Wen-Liang Zhou, Anna Moore, Shaina Short, and Katerina Oikonomou. The decade of the dendritic nmda spike. 88:2991–3001, 11 2010.

[29] Graeme W. Davis. Homeostatic control of neural activity: From phenomenology to molecular design. *Annual Review of Neuroscience*, 29(1):307–323, 2006. doi: 10.1146/annurev.neuro.28.061604.135751. URL https://doi.org/10.1146/annurev.neuro.28.061604.135751. PMID: 16776588.

[30] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole, 8th edition, January 2011. ISBN-13: 978-0-538-73352-6.

[31] James J. Dicarlo, Davide Zoccolan, and Nicole C. Rust. How does the brain solve visual object recognition? *Neuron*, 73(3):415434, 2012. doi: 10.1016/j.neuron.2012.01.010.

[32] Pedro Domingos. *The master algorithm: how the quest for the ultimate learning machine will remake our world*. Penguin Books, 2017.

[33] D. E. Erb and J. T. Povlishock. Neuroplasticity following traumatic brain injury: a study of gabaergic terminal loss and recovery in the cat dorsal lateral vestibular nucleus. *Experimental Brain Research*, 83(2):253–267, Jan 1991. ISSN 1432-1106. doi: 10.1007/BF00231151. URL https://doi.org/10.1007/BF00231151.

[34] Michael R. Ferrier. Toward a universal cortical algorithm: Examining hierarchical temporal memory in light of frontal cortical function. *CoRR*, abs/1411.4702, 2014. URL http://arxiv.org/abs/1411.4702.

[35] David A. Ferrucci. Ibm's watson/deepqa. *SIGARCH Comput. Archit. News*, 39 (3):–, June 2011. ISSN 0163-5964. doi: 10.1145/2024723.2019525. URL http://doi.acm.org/10.1145/2024723.2019525.

[36] David A. Ferrucci. Ibm's watson/deepqa. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages –, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0472-6. URL http://dl.acm.org/citation.cfm?id=2000064.2019525.

[37] David J. Field. Relations between the statistics of natural images and the response properties of cortical cells. *J. Opt. Soc. Am. A*, 4(12):2379–2394, Dec 1987. doi: 10.1364/JOSAA.4.002379. URL http://josaa.osa.org/abstract.cfm?URI=josaa-4-12-2379.

[38] David J. Field. What is the goal of sensory coding? *Neural Comput.*, 6(4):559–601, July 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.4.559. URL http://dx.doi.org/10.1162/neco.1994.6.4.559.

[39] Mark Fischetti. Computers versus brains, Nov 2011. URL https://www.scientificamerican.com/article/computers-vs-brains/.

[40] P. Gaikwad, A. Mandal, P. Ruth, G. Juve, D. Krl, and E. Deelman. Anomaly detection for scientific workflow applications on networked clouds. In *2016 International Conference on High Performance Computing Simulation (HPCS)*, pages 645–652, July 2016. doi: 10.1109/HPCSim.2016.7568396.

[41] Alex Graves, Abdel-Rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013. doi: 10.1109/icassp.2013.6638947.

[42] Larry Greenemeier. Will ibm's watson usher in a new era of cognitive computing?, Nov 2013. URL https://www.scientificamerican.com/article/will-ibms-watson-usher-in-cognitive-computing/.

[43] Tingting Han, Hongxun Yao, Xiaoshuai Sun, Sicheng Zhao, and Yanhao Zhang. Unsupervised discovery of crowd activities by saliency-based clustering. *Neurocomput.*, 171(C):347–361, January 2016. ISSN 0925-2312. doi: 10.1016/j.neucom.2015.06.048. URL https://doi.org/10.1016/j.neucom.2015.06.048.

[44] Kai Häussermann, Oliver Zweigle, and Paul Levi. A novel framework for anomaly detection of robot behaviors. *Journal of Intelligent & Robotic Systems*, 77(2):361–375, Feb 2015. ISSN 1573-0409. doi: 10.1007/s10846-013-0014-5. URL https://doi.org/10.1007/s10846-013-0014-5.

[45] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin. Biological and machine intelligence (bami). Initial online release 0.4, 2016. URL http://numenta.com/biological-and-machine-intelligence/.

[46] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, 10, 2016. doi: 10.3389/fncir.2016.00023.

[47] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Times Books, 2004. ISBN 0805074562.

[48] Jordan Hochenbaum, Owen S. Vallis, and Arun Kejariwal. Automatic anomaly detection in the cloud via statistical learning. *CoRR*, abs/1704.07706, 2017. URL http://arxiv.org/abs/1704.07706.

[49] Jeremy Hsu. Scientists work on artificial cat brain, Apr 2010. URL http://www.nbcnews.com/id/36872308/ns/technology_and_science-science/t/scientists-work-artificial-cat-brain/#.WgtOxlVKuUk.

[50] htm community. htm-community/comportex, Sep 2016. URL https://github.com/htm-community/comportex.

[51] R. J. Hyndman. Time series data library, Jul 2010. URL https://robjhyndman.com/TSDL/.

[52] JeffryS. Isaacson and Massimo Scanziani. How inhibition shapes cortical activity. *Neuron*, 72(2):231 – 243, 2011. ISSN 0896-6273. doi: https://doi.org/10.1016/

j.neuron.2011.09.027. URL http://www.sciencedirect.com/science/article/pii/S0896627311008798.

[53] Vladislav Ishimtsev and Evgeny Burnaev. numenta/nab. URL https://github.com/numenta/NAB/tree/master/nab/detectors/knncad.

[54] Eduardo J. Spinosa and Joo Gama. Olindda: A cluster-based approach for detecting novelty and concept drift in data streams. pages 448–452, 01 2007.

[55] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1999. ISBN 0849371813.

[56] J. H. Kaas. Neocortex in early mammals and its subsequent variations. *Ann. N. Y. Acad. Sci.*, 1225:28–36, Apr 2011.

[57] Jon H Kaas. Topographic maps are fundamental to sensory processing. *Brain Research Bulletin*, 44(2):107 – 112, 1997. ISSN 0361-9230. doi: https://doi.org/10.1016/S0361-9230(97)00094-4. URL http://www.sciencedirect.com/science/article/pii/S0361923097000944.

[58] E. Keogh, J. Lin, and A. Fu. Hot sax: efficiently finding the most unusual time series subsequence. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–, Nov 2005. doi: 10.1109/ICDM.2005.79.

[59] Gift Khangamwa. Detecting network intrusions using hierarchical temporal memory. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering E-Infrastuctures and E-Services for Developing Countries*, page 4148, 2011. doi: 10.1007/978-3-642-23828-4_5.

[60] Victor AF Lamme, Hans Supr, and Henk Spekreijse. Feedforward, horizontal, and feedback processing in the visual cortex. *Current Opinion in Neurobiology*, 8(4):529 – 535, 1998. ISSN 0959-4388. doi: https://doi.org/10.1016/S0959-4388(98)80042-1. URL http://www.sciencedirect.com/science/article/pii/S0959438898800421.

[61] N. Laptev and S. Amizadeh. S5 - a labeled anomaly detection dataset, version 1.0, 2015. URL https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70.

[62] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining, pages 1939–1947. ACM, 2015.

[63] M. E. Larkum, J. J. Zhu, and B. Sakmann. A new cellular mechanism for coupling inputs arriving at different cortical layers. *Nature*, 398(6725):338–341, Mar 1999.

[64] Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015. doi: 10.1109/icmla.2015.141.

[65] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436444, 2015. doi: 10.1038/nature14539.

[66] Seungmin Lee, Gisung Kim, and Sehun Kim. Self-adaptive and dynamic clustering for online anomaly detection. *Expert Syst. Appl.*, 38(12):14891–14898, November 2011. ISSN 0957-4174. doi: 10.1016/j.eswa.2011.05.058. URL http://dx.doi.org/10.1016/j.eswa.2011.05.058.

[67] Matthew Lewis and Robert Wiseman. Arterial pressure dataset. Mar 2018.

[68] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[69] Books LLC. *Denial-of-Service Attacks: Sql Slammer, Denial-of-Service Attack, Smurf Attack, Fork Bomb, Operation Titstorm, Zombie Computer, Slashdot Effect*. General Books LLC, 2010. ISBN 9781155738659. URL https://books.google.com/books?id=DPiYSQAACAAJ.

[70] Simona Lodato and Paola Arlotta. Generating neuronal diversity in the mammalian cerebral cortex. *Annual Review of Cell and Developmental Biology*, 31(1):699–720, 2015. doi: 10.1146/annurev-cellbio-100814-125353. URL https://doi.org/10.1146/annurev-cellbio-100814-125353. PMID: 26359774.

[71] Attila Losonczy, Judit K. Makara, and Jeffrey C Magee. Compartmentalized dendritic plasticity and input feature storage in neurons. *Nature*, 452 7186:436–41, 2008.

[72] Wei Lu and Issa Traore. A new unsupervised anomaly detection framework for detecting network attacks in real-time. In *Proceedings of the 4th International Conference on Cryptology and Network Security*, CANS'05, pages 96–109, Berlin,

Heidelberg, 2005. Springer-Verlag. ISBN 3-540-30849-0, 978-3-540-30849-2. doi: 10.1007/11599371_9. URL http://dx.doi.org/10.1007/11599371_9.

[73] JanH. Lui, DavidV. Hansen, and ArnoldR. Kriegstein. Development and evolution of the human neocortex. *Cell*, 146(1):18 – 36, 2011. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2011.06.030. URL http://www.sciencedirect.com/science/article/pii/S0092867411007057.

[74] F.Y. Edgeworth M.A. Xli. on discordant observations. *Philosophical Magazine*, 23 (143):364–375, 1887. doi: 10.1080/14786448708628471. URL http://dx.doi.org/10.1080/14786448708628471.

[75] Amogh Mahapatra, Nisheeth Srivastava, and Jaideep Srivastava. Contextual anomaly detection in text data. *Algorithms*, 5(4):469–489, 2012. ISSN 1999-4893. doi: 10.3390/a5040469. URL http://www.mdpi.com/1999-4893/5/4/469.

[76] G. Major, M. E. Larkum, and J. Schiller. Active properties of neocortical pyramidal neuron dendrites. *Annu. Rev. Neurosci.*, 36:1–24, Jul 2013.

[77] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. 04 2015.

[78] Markos Markou and Sameer Singh. A neural network-based novelty detector for image sequence analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 28:1664–1677, 2006. ISSN 0162-8828. doi: doi.ieeecomputersociety.org/10.1109/TPAMI.2006.196.

[79] Henry Markram, Karlheinz Meier, Thomas Lippert, Sten Grillner, Richard Frackowiak, Stanislas Dehaene, Alois Knoll, Haim Sompolinsky, Kris Verstreken, Javier DeFelipe, Seth Grant, Jean-Pierre Changeux, and Alois Saria. Introducing the human brain project. *Procedia Computer Science*, 7(Supplement C):39 – 42, 2011. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2011.12.015. URL http://www.sciencedirect.com/science/article/pii/S1877050911006806. Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11).

[80] K. A. C. Martin and S. Schroder. Functional heterogeneity in neighboring neurons of cat primary visual cortex in response to both artificial and natural stimuli. *Journal of Neuroscience*, 33(17):73257344, 2013. doi: 10.1523/jneurosci.4071-12.2013.

[81] Mind Matters. Are whales smarter than we are?, Jan 2008. URL https://blogs.scientificamerican.com/news-blog/are-whales-smarter-than-we-are/.

[82] Mu ming Poo, Jiu lin Du, NancyY. Ip, Zhi-Qi Xiong, Bo Xu, and Tieniu Tan. China brain project: Basic neuroscience, brain diseases, and brain-inspired computing. *Neuron*, 92(3):591 – 596, 2016. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2016.10.050. URL http://www.sciencedirect.com/science/article/pii/S0896627316308005.

[83] James Mnatzaganian, Ernest Fokoué, and Dhireesha Kudithipudi. A mathematical formalization of hierarchical temporal memory cortical learning algorithm's spatial pooler. *CoRR*, abs/1601.06116, 2016.

[84] James H. Moor. Turing test. In *Encyclopedia of Computer Science*, pages 1801–1802. John Wiley and Sons Ltd., Chichester, UK. ISBN 0-470-86412-5. URL http://dl.acm.org/citation.cfm?id=1074100.1074882.

[85] Vernon B. Mountcastle. An organizing principle for cerebral function: The unit model and the distributed system. In Gerald M. Edelman and Vernon V. Mountcastle, editors, *The Mindful Brain*, pages 7–50. MIT Press, Cambridge, MA, 1978.

[86] Maria Moustra, Marios Avraamides, and Chris Christodoulou. Artificial neural networks for earthquake prediction using time series magnitude data or seismic electric signals. *Expert Syst. Appl.*, 38(12):15032–15039, November 2011. ISSN 0957-4174. doi: 10.1016/j.eswa.2011.05.043. URL http://dx.doi.org/10.1016/j.eswa.2011.05.043.

[87] Trung Thanh Nguyen, Anh Tuan Nguyen, Tuan Anh Ha Nguyen, Ly Thi Vu, Quang Uy Nguyen, and Long Dao Hai. Unsupervised anomaly detection in online game. In *Proceedings of the Sixth International Symposium on Information and Communication Technology*, SoICT 2015, pages 4–10, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3843-1. doi: 10.1145/2833258.2833305. URL http://doi.acm.org/10.1145/2833258.2833305.

[88] C.R. Noback, N.L. Strominger, R.J. Demarest, and D.A. Ruggiero. *The Human Nervous System: Structure and Function*. Number no. 744; no. 2005 in SpringerLink: Springer e-Books. Humana Press, 2005. ISBN 9781588290403. URL https://books.google.com/books?id=UnRO3A_cS44C.

[89] Randolph Nudo. Recovery after brain injury: Mechanisms and principles. 7:887, 12 2013.

[90] Numenta. Nab competition 2016 winners, . URL https://numenta.com/blog/2016/08/10/numenta-anomaly-benchmark-nab-competition-2016-winners/.

[91] Numenta. Numenta.com, . URL https://numenta.com/.

[92] Numenta. numenta/htm.java, Sep 2017. URL https://github.com/numenta/htm.java.

[93] The Editors of Encyclopdia Britannica. Hindbrain, Jul 2015. URL https://www.britannica.com/science/hindbrain.

[94] T. Otsuka, Y. Torii, and T. Ito. Anomaly detection algorithm for localized abnormal weather using low-cost wireless sensor nodes. In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pages 304–308, Nov 2014. doi: 10.1109/SOCA.2014.34.

[95] Clyde W. Oyster. *The human eye: structure and function*. Sinauer Associates, 2006.

[96] E. H. M. Pena, M. V. O. de Assis, and M. L. Proena. Anomaly detection using forecasting methods arima and hwds. In *2013 32nd International Conference of the Chilean Computer Science Society (SCCC)*, pages 63–66, Nov 2013. doi: 10.1109/SCCC.2013.18.

[97] Srinath Perera. Introduction to anomaly detection: Concepts and techniques, Feb 2016. URL https://iwringer.wordpress.com/2015/11/17/anomaly-detection-concepts-and-techniques/.

[98] L. Petreanu, T. Mao, S. M. Sternson, and K. Svoboda. The subcellular organization of neocortical excitatory connections. *Nature*, 457(7233):1142–1145, Feb 2009.

[99] Mario Alfonso Prado-Romero, Christian Doerr, and Andres Gago-Alonso. Discovering bitcoin mixing using anomaly detection. In *Iberoamerican Congress on Pattern Recognition (CIARP)*, 2017.

[100] David C. Preston and Barbara Shapiro. Comte. *Electromyography and Neuromuscular Disorders*. Elsevier Health Sciences, 2013.

[101] Kevin L. Priddy and Paul E. Keller. *Artificial Neural Networks: An Introduction (SPIE Tutorial Texts in Optical Engineering, Vol. TT68)*. SPIE- International Society for Optical Engineering, 2005. ISBN 0819459879.

[102] Afrooz Purarjomandlangrudi, Amir Hossein Ghapanchi, and Mohammad Esmalifalak. A data mining approach for fault diagnosis: An application of anomaly detection algorithm. *Measurement*, 55(Supplement C):343 – 352, 2014. ISSN 0263-2241. doi: https://doi.org/10.1016/j.measurement.2014.05.029. URL http://www.sciencedirect.com/science/article/pii/S0263224114002504.

[103] Scott Purdy. Encoding data for HTM systems. *CoRR*, abs/1602.05925, 2016. URL http://arxiv.org/abs/1602.05925.

[104] John A. Quinn and Masashi Sugiyama. A least-squares approach to anomaly detection in static and sequential data. *Pattern Recognition Letters*, 40(Supplement C):36 – 40, 2014. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2013.12.016. URL http://www.sciencedirect.com/science/article/pii/S016786551300514X.

[105] J. C. Rah, E. Bas, J. Colonell, Y. Mishchenko, B. Karsh, R. D. Fetter, E. W. Myers, D. B. Chklovskii, K. Svoboda, T. D. Harris, and J. T. Isaac. Thalamocortical input onto layer 5 pyramidal neurons measured using quantitative large-scale array tomography. *Front Neural Circuits*, 7:177, 2013.

[106] Srikanth Ramaswamy and Henry Markram. Anatomy and physiology of the thick-tufted layer 5 pyramidal neuron. *Frontiers in Cellular Neuroscience*, 9, 2015. doi: 10.3389/fncel.2015.00233.

[107] E. J. Robinson. Young childrens detection of semantic anomaly response and judgement task. *PsycTESTS Dataset*, 1992. doi: 10.1037/t42214-000.

[108] E . T . Rolls and Martin J. Tovee. Sparseness of the neuronal representation of stimuli in the primate temporal visual cortex. *Journal of neurophysiology*, 73 2: 713–26, 1995.

[109] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[110] Bernard Rosner. Percentage points for a generalized esd many-outlier procedure. *Technometrics*, 25(2):165, 1983. doi: 10.2307/1268549.

[111] Saharon Rosset and Aron Inger. Kdd-cup 99: Knowledge discovery in a charitable organization's donor database. *SIGKDD Explor. Newsl.*, 1(2):85–90, January 2000. ISSN 1931-0145. doi: 10.1145/846183.846204. URL http://doi.acm.org/10.1145/846183.846204.

[112] J. Schiller and Y. Schiller. NMDA receptor-mediated dendritic spikes and coincident signal amplification. *Curr. Opin. Neurobiol.*, 11(3):343–348, Jun 2001.

[113] J. Schiller, G. Major, H. J. Koester, and Y. Schiller. NMDA spikes in basal dendrites of cortical pyramidal neurons. *Nature*, 404(6775):285–289, Mar 2000.

[114] Markus Schneider, Wolfgang Ertel, and Fabio T. Ramos. Expected similarity estimation for large-scale batch and streaming anomaly detection. *CoRR*, abs/1601.06602, 2016. URL http://arxiv.org/abs/1601.06602.

[115] N. Singh and C. Olinsky. Demystifying numenta anomaly benchmark. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1570–1577, May 2017. doi: 10.1109/IJCNN.2017.7966038.

[116] Nelson Spruston. Spruston n. pyramidal neurons: dendritic structure and synaptic integration. nat rev neurosci 9: 206-221. 9:206–21, 04 2008.

[117] Rand S Swenson. Chapter 11 - the cerebral cortex. URL https://www.dartmouth.edu/~rswenson/NeuroSci/chapter_11.html.

[118] Maciej Szmit and Anna Szmit. Usage of modified holt-winters method in the anomaly detection of network traffic: Case studies. 5, 05 2012.

[119] L. Tarassenko. Novelty detection for the identification of masses in mammograms. *4th International Conference on Artificial Neural Networks*, 1995. doi: 10.1049/cp:19950597.

[120] Matthew Taylor, Scott Purdy, breznak, Chetan Surpur, Austin Marshall, David Ragazzi, Subutai Ahmad, numenta ci, Andrew Malta, Pascal Weinberger, Akhila, Marcus Lewis, Richard Crowder, Marion Le Borgne, Yuwei, Christopher Simons, Ryan J. McCall, Mihail Eric, Utensil Song, keithcom, Nathanael Romano, Sagan Bolliger, vitaly krugl, hernandezurbina, James Bridgewater, Ian Danforth, Jared Weiss, Tom Silver, David Ray, and Luiz Scheinkman. numenta/nupic: 1.0.3, September 2017. URL https://doi.org/10.5281/zenodo.891005.

[121] Twitter. twitter/anomalydetection, Aug 2015. URL https://github.com/twitter/AnomalyDetection.

[122] S. B. Udin and J. W. Fawcett. Formation of topographic maps. *Annual Review of Neuroscience*, 11(1):289–327, 1988. doi: 10.1146/annurev.ne.11.030188.001445. URL https://doi.org/10.1146/annurev.ne.11.030188.001445. PMID: 3284443.

[123] William E. Vinje and Jack L. Gallant. Sparse coding and decorrelation in primary visual cortex during natural vision. *Science*, 2000.

[124] L von Melchner, S L Pallas, and M Sur. Visual behaviour mediated by retinal projections directed to the auditory pathway. *Nature*, 404:871–876, 2000.

[125] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan. Statistical techniques for online anomaly detection in data centers. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 385–392, May 2011. doi: 10.1109/INM.2011.5990537.

[126] D.B. Webster, A.N. Popper, and R.R. Fay. *The Mammalian auditory pathway: neuroanatomy*. Springer handbook of auditory research. Springer-Verlag, 1992. ISBN 9780387976785. URL https://books.google.com/books?id=x6HwAAAAMAAJ.

[127] Jeffrey Wong, Chris Colburn, Elijah Meeks, and Shankar Vedaraman. Radoutlier detection on big data, Feb 2015. URL https://medium.com/netflix-techblog/rad-outlier-detection-on-big-data-d6b0494371cc.

[128] Shih-Cheng Yen, Jonathan Baker, and Charles M Gray. Heterogeneity in the responses of adjacent neurons to natural stimuli in cat striate cortex. *J Neurophysiol*, 97(2):13261341, 2007. doi: http://dx.doi.org/10.1152/jn.00747.2006. URL http://dx.doi.org/10.1152/jn.00747.2006.

[129] Yumiko Yoshimura, Hiromichi Sato, Kazuyuki Imamura, and Yasuyoshi Watanabe. Properties of horizontal and vertical inputs to pyramidal cells in the superficial layers of the cat visual cortex. *Journal of Neuroscience*, 20(5):1931–1940, 2000. ISSN 0270-6474. URL http://www.jneurosci.org/content/20/5/1931.

[130] Yingbing Yu. A survey of anomaly intrusion detection techniques. *J. Comput. Sci. Coll.*, 28(1):9–17, October 2012. ISSN 1937-4771. URL http://dl.acm.org/citation.cfm?id=2379703.2379707.

[131] Z. G. Zhou and P. Tang. Improving time series anomaly detection based on exponentially weighted moving average (ewma) of season-trend model residuals. In *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3414–3417, July 2016. doi: 10.1109/IGARSS.2016.7729882.

[132] Karen Zito and Karel Svoboda. Activity-dependent synaptogenesis in the adult mammalian cortex. *Neuron*, 35(6):10151017, 2002. doi: 10.1016/s0896-6273(02)00903-0.